Ultra-Dependable, High-Performance, Real-Time Signal Processing

Jien-Chung Lo, Donald W. Tufts and James W. Cooley
Department of Electrical and Computer Engineering
The University of Rhode Island
Kingston, RI 02881-0805

Abstract

The Active Nodal Task Seeking (ANTS) approach to the design of multicomputer systems is named for its basic component: an Active Nodal Task-Seeker (ANT). In this system, there is no load balancing or load sharing, instead, each ANT computing node is actively finding out how it can contribute to the execution of the needed tasks. A run-time partition is established such that some of the ANT computing nodes are under exhaustive diagnosis at any given time. An ANTS multicomputer system can achieve a mean time to failure of more than 20 years with just 8 computing nodes and 3 buses, while the minimum requirements are 3 computing nodes and 1 bus, and with a worst case computing node failure rate of 5×10^{-4} per hour.

In this final report, we present the many findings during the project duration. First, the first version of ANTS kernel has been implemented on a network of UNIX workstations. Employing TCP/IP protocol for communication via Ethernet LAN. The second version of ANTS kernel will exist in a workstation in the form of a UNIX daemon. In such a case, there will be no performance shortfall observable by the casual workstation user when the kernel is not engaged in any ANTS job. Based on these implementations, we have successfully demonstrated the proposed fault tolerant techniques via fault injection experiments. The system detects faults and recovery itself as expected within seconds. The dependability goal predicted in the theoretical model is thus validated. We then program and execute several well-known algorithms in ANTS distributed computing environment to demonstrate the high-performance features. Algorithm designs for ANTS environment are also discussed.

This work has been motivated by the need to develop high-performance multicomputer systems for radar, active and passive sonar, and electronic warfare that can provide ultra-dependable performance for more than 20 years without field repairs. We argue that high performance is also an attribute of an ANTS computing system, because the overhead of dynamic task scheduling is reduced and because efficient use is made of the available processing resources. We have successfully demonstrated that both dependability and high-performance goals of our proposal have been achieved. This forms a basis for a practical real-time systems.

This work has been supported by the Office of Naval Research under grant N00014-94-1-0479.

1 Introduction

Active Nodal Task Seeking (ANTS) [1] is an approach to the design of high-performance, ultra-dependable multicomputer systems. The goal is to design an ultra-dependable, real-time, and distributed system for computationally intensive signal and data processing applications. The standard for ultra-dependability is set at a mean time to failure (MTTF) exceeding 20 years. The hardware organization of ANTS is a distributed system with multiple-bus connected stand-alone computing nodes. This system uses a novel operating mode to achieve the ultra-dependability and maintain a high degree of computational performance.

The only direct predecessor to ANTS is the now-forgotten, but very successful, Safeguard multiprocessor system of Bell Telephone Laboratories. Before reading on or looking at our list of references, we urge the reader to search his or her memory for information about the existence or architecture of the Safeguard system. In books on the design of computer systems [2, 3, 4] no references to or mention of Safeguard appears. Safeguard [5] was the high-performance, dependable multiprocessor for computation and control of the Safeguard anti-ballistic missile (ABM) defense system of the early 1970's. ANTS extends and generalizes the approach of Safeguard, and we intend to provide performance and dependability analysis for ANTS multicomputer systems. Two other predecessor systems are FTMP [6] and SIFT [7]. Unlike ANTS, these systems use at least three times the resources required by the application. Triads of processors are assigned to execute task segments. Also FTMP adopted a fully synchronous approach which uses a hardware implemented bit-by-bit voting on all transactions. The asynchronous nature of ANTS is closer to that of SIFT.

The term dependability collectively describes the common fault-tolerant system measurements, such as reliability, availability, MTTF, *etc.*. Depending on the mission, one or more of these measurements are used in the system specification [8]. For instance, electronic switching systems (ESSs) are designed to achieve high availability [9]. Avionics control systems, such as FTMP and SIFT are designed to achieve ultra-high reliability for a short mission time. For military and space-bound applications, a long MTTF is required to ensure the operation under severe circumstances [8].

Distributed systems are potentially fault-tolerant. However, utilizing this intrinsic capability for fault-tolerance is not a straightforward task. Many existing dependable systems

still use special hardware designs to achieve the desired level of dependability. The Advanced Architecture On-board Processor by Harris Corporation [10, 11] uses self-checking RISC processors and chordal ring. The Advanced Automation System by IBM [12, 13] needs redundant components at each subsystem due to its wide area distribution nature. The on-board computer of the Japanese satellite Hiten [14] uses stepwise negotiating voting, which is a combination of mutual checking by data comparison and self-checking.

Special hardware components may not be as necessary if an appropriate software approach is applied. In Delta-4 [15], active replication of software programs is used to ensure fault-tolerance. A similar strategy is used in Manetho [16], where each application process is replicated by a set named: troupe. These systems are distributed systems with no special hardware designed for fault-tolerance. Unfortunately, replication of programs also means a significant reduction in system computational performance.

Existing dependable multicomputer or multiprocessor designs assume that the computing nodes or the processors are passive in their operating mode. Load balancing or load sharing [17] is thus required because the idle computing nodes remain idle until new jobs are distributed to them. The busy nodes are responsible for distributing tasks to the idle ones.

From the fault tolerance perspective, active nodal task seeking (ANTS) offers several advantages:

- On-line error checking is simple and straightforward. The two extreme failure modes: *fail-silent* and *fail-uncontrolled* [15] are easily detected since the failed computing node will not be actively seeking a new task in a fixed time frame.

- Fault tolerance can be attained without seriously degrading the system performance. There is no need for special hardware or replication of programs in ANTS and thus no serious performance degradation. If ultra-reliability is required during the critical phase of the mission, jobs are triplicated in an asynchronous, distributed mode. In this case, the voting function is implemented in software similar to that in SIFT [7].

- There is no need for synchronization of nodes or programs. The active nodal task-seeker (ANT) computing nodes operate asynchronously and independently. Of course, they must work cooperatively. Each ANT node, when seeking a new task, must be

aware of the pertinent work that has been done by itself and other ANT nodes as it is posted in the bulletin board (task tables). Each ANT node keeps its own copy of the task table. The entries of these distributed task tables are updated via broadcasting of task acquiring and task completion messages over the communication network.

- The ANTS system is designed such that no single dependability-critical component exists in the system. Error checking functions are handled distributively. Near coincident failures or even multiple failures can only degrade the system performance temporarily but cannot paralyze the continuous operation of an ANTS system.

The ANTS concept is efficient since only little time and little computational resources are used for scheduling tasks on the computing nodes. Therefore, computing nodes can do more work on the real-time applications in any fixed time interval. Of course, this requires an extra, one-time effort when developing and programming the real-time tasks. The ANTS concept is effective because the system throughput is close to that of an ideal system in which each processor that is not under repair knows exactly how to contribute to the current set of real-time application tasks or testing tasks.

Although ANTS concept applies to many existing distributed computing architectures, in this project, we concentrate on network of workstations (NOWs) configuration. The main reason is the accessibility of such a configuration in the academic setting. Also, a network of workstations can be commonly found in many work places including ours. Researchers have argued that the potential aggregated computational power of these workstations should be used for coarse grained parallel computations of large scale problems. There have been many works on this subject.

2 Previous Works

Distributed computing is potentially dependable since there are built-in redundancies and the processing units are distributed. However, we would like to point out that none of these existing systems, both theoretical and experimental, can efficiently achieve both high-performance and ultra-dependability at the same time [10, 11, 12, 13, 14, 15, 16]. There are numerous reasons, but, the one we believe is the main cause is that most known distributed

systems are not really *distributed.* By this we mean that there is always an implied relationship between active processing units and passive processing units. This active/passive relationship gives rise to the load balancing problem [18]. For real-time radar, active sonar, and electronic warfare systems applications, we need to guarantee that sufficient computational power will be available, even in the presence of component failures, and can be allocated to compute all high priority tasks with the real-time deadlines.

The ANTS concept is active, asynchronous and adaptive in its operations. The first active mode distributed computing system that we are aware of is the Bell Laboratories' Safeguard [5]. Developed in 1970's for the Anti-Ballistic Missile (ABM) system, Safeguard has many innovative features that we have incorporated into the ANTS concept. The most notable feature is the fully distributed task scheduling. The computing nodes in Safeguard find their own task segments without any centralized mechanism. This feature is retained in ANTS [1] to facilitate dependable, high performance computations.

The IBM AS/400 [19] is a shared memory distributed computing system. This system uses active mode operation plus special registers to guarantee an automatic load balancing. The shared main memory space is the centralized mechanism that will later become a dependability issue as well as a performance bottleneck. Piranha [20] has been realized in a network of workstations and on CM-5 as well. Piranha system assumed a centralized mechanism for job distribution in that the task queue which consists of all waiting tasks is in one centralized location. This obviously is a serious dependability issue and eventually would result in a performance bottleneck. When many workstations are included in the computation of large scale problems, all communication to that one centralized location, for accessing the task queue, can certainly create a bottleneck. Phish [21] uses two levels of centralized job distribution mechanisms. At its top level, Phish assumed that each computing node will make inquiry to the centralized job queue in an interval of 30 seconds. This will greatly reduce the effect of long communication delays and avoid possible bottlenecks. Once a workstation has gained control over a job from the centralized job queue, it creates a run-time *clearing house* for dispatching tasks to other workstations. This clearing house represents yet another centralized control mechanism. Also, this lower level scheduling policy automatically rules out the possibility of contribution from some workstations to a given job, since the domain of participation is defined by the clearing house.

All these three active mode concepts are far from being considered dependable. The IBM AS/400 and Piranha depend on the inherent redundancy of any parallel or distributed computing system to provide potential dependability. The advocates of Phish [21] claim that the fault-tolerance is guaranteed since redundant copies of a job exist in both top level and lower level centralized mechanism. However, the fact remains that no specific fault tolerant techniques have been proposed for these machines. On the other hand, many fault-tolerant distributed or parallel systems suffer significant performance loss due to the inclusion of redundant fault tolerance capabilities.

Recently, some similar works have been found in the public domain. These are freely available packages which enable a programmer to execute programs across workstations on a network. The World Wide Web addresses of these packages are:

PVM: Parallel Virtual Machine at http://www.epm.ornl.gov/pvm/

LAM: Local Area Multicomputer at http://www.osc.edu/lam.html

CHIMP at http://www.epcc.ed.ac.uk/epcc-projects/CHIMP/index.html

MPICH at http://www.mcs.anl.gov/home/lusk/mpich

AMOEBA at http://www.am.cs.vu.nl/amoeba.html

All these above implementations are operated in a traditional manner where a centralized mechanism is assumed not only for the distribution of jobs and tasks, but also for tick-by-tick operations. Therefore, our earlier criticism of traditional system not being truly distributed also applies here. Further, none of these packages is designed with dependability in mind.

The ANTS concept was developed [1] to achieve both dependability and high performance. We assume that each computing node or workstation maintains its own task queue of all awaiting tasks in the system. New tasks can be spawned easily by adding them to the task queue. The coherency of the distributed task queues are maintained by means of inter-node communications. While these scheduling methods ensure high-performance, they also provide enough redundancy for dependable computing. Simple error detecting and recovering techniques can be easily incorporated into the distributed kernel with minimum performance loss. The experimental results will show that a near linear speedup is likely

when the average task execution time is much larger than the average communication delay. We will further show that the inclusion of fault tolerant techniques accounts for less than 5% performance loss.

3 An Overview of ANTS Multicomputer System

Figure 1 shows the underlying hardware configuration of this study: a network of workstations. Specifically, we implemented the ANTS concept on UNIX workstations that are interconnected via a 10Mb/s Ethernet. It is worth noting that this is merely one of the many possible hardware configurations for our ANTS concept.

In an ANTS system, an idle computing node will seek out the information it needs about pending, high priority work from partly prescribed, partly updated task tables. This it does instead of waiting for other computing nodes to send it a task to perform. Naturally, predecessor/successor conditions like those in a data flow architecture must be checked by the node. The advantage of this concept is that a true distributed system is guaranteed, wherein all programs, including operating system diagnostics and applications, can be executed distributively.

We define the term high-performance as the capability to approximately achieve the potential computational power. This is a suitable definition for a scalable environment where computing nodes can be added to gain higher performance. ANTS can adapt itself to accommodate the increase in the number of computing nodes without any hardware or software modification. For the fault-tolerant version, the ANTS can add or drop computing nodes during run-time without adversely affecting the jobs that are currently running. In a locally connected system such as a system with several workstations interconnected via a local area network, this attribute of ANTS is very useful. Additional workstations can be added to obtain higher performance. Also, during run-time, some workstations that are heavily loaded with local non-ANTS jobs will naturally not pick any ANTS jobs. When the load on these workstations go down, these workstations can easily re-enter the ANTS operation by again actively seeking out the next available tasks. Task allocation and load balancing are thus done dynamically.

3.1 Job Distribution and Task Queues

When an ANTS job initiates, all related information such as tasks and data dependencies are broadcast to all ANTS computing nodes. A computing node, upon receiving the information, will update its own local task queue for this new job. A computing node, when idle, will pick up the high priority task from its own task queue for execution. There are several criteria for the task selection:

1. In a real-time environment, the priority associated with each task will be considered first. Priorities of tasks are updated regularly according to the present deadline situation.

2. The second criterion is for an ANTS node to pick a task whose predecessor process, or at least one of them, has been executed by the same node. By doing so, we avoid the additional data transfer time since the predecessor data, or at least part of the needed data, is available locally.

3. A task queue may consists of thousands of awaiting tasks. Searching though all entries for an optimal match may be time consuming. Therefore, a user can define a lookahead threshold which limits the depth of search by the kernel. The search depth usually directly relates to the algorithm itself.

4. The relative computational capabilities of individual nodes in a network of heterogeneous nodes, also determines the selection of tasks by a given node.

3.2 Conflict and Coherency

A centralized task distribution can avoid the problem of conflict and coherency since the centralized mechanism handles requests sequentially. In ANTS, we need to address these problems due to the distributed nature of the task tables. A conflict may occur when two or more computing nodes pick up the same task for execution. Each computing node will broadcast the message claiming the ownership of the task and then will start the execution immediately after. When other nodes receive the message, they will realize that they have picked the same task. We use a simple time-stamp method to help resolve the conflict. A

time-stamp is associated with each task ownership message. By comparing the time-stamps, a computing node will decide whether to continue the execution or to abandon the execution. In the worst case, a nearly completed task may be abandoned.

3.3 Communication Delays

Even though we consider coarse grain applications for a network of workstations, the communication delays still play an important role. For example, in Piranha [20], a task duration of 1 second is chosen for a reasonable speed up curve. In our initial implementation, we also find that a long average execution time for tasks enables a near linear speed-up curve. Of course, the numerical answer to what is the proper average execution time of tasks is directly related to the average communication delay of the underlying network. Nonetheless, we may conclude that it is beneficial to reduce the frequency of communication. We note that in ANTS all communications are essentially broadcast messages, but, in an Ethernet this is equivalent to point-to-point communication.

4 Dependability Designs

This section addresses the design features of ANTS which are aimed at enhancing the dependability. An analytical model is developed to evaluate the mean time to failure (MTTF) of the ANTS system. The design goal is set at achieving an MTTF of 20 years and more. We show that a simple combination of 8 computing nodes and 3 buses, with minimum requirement of 3 computing nodes and one bus, could achieve this goal, assuming that the computing node failure rate is 5×10^{-4} per hour, or a MTTF of 2,000 hours. This means that commercially available components are suitable for constructing an ultra-dependable ANTS system. Also the specialized applications for which ANTS is intended, such as a digital-cellular-radio based station, a central-office digital-communications signal processing interface, an active/passive sonar system, or a radar system naturally lend themselves to efficient and effective reasonableness checks. Error detection/correction codes provide such checks in communication systems. The physical consistency of estimated parameter values provide such checks in sonar and radar .

4.1 Run Time Partitioning

The computing nodes, as well as the buses, are partitioned into three groups during run time: the up-and-running green group, the stop-and-checking yellow group and the red group for removed faulty nodes and buses. The number of computing nodes allocated for the green group must always be greater than the minimum requirement for execution of the applications at any given time. The rest of the computing nodes are then in the yellow group unless they have been found to fail, in which case they are removed to the stopped, red group. The grouping is based on the status of the computing nodes and/or buses. Therefore, there is no visible physical grouping, and no need for special hardware design for switching between the two run-time groups.

Since each computing node seeks work for itself, placing a computing node in the yellow, checking group is accomplished by making the diagnostic program the highest priority task for it, through a timer for example. To exercise the bus, one of the yellow computing nodes will gain exclusive access to one of the buses and check its vitality. In other words, there are two types of diagnostic programs: one checks computing node only and the other checks both a computing node and a bus.

Besides the diagnostic and other bookkeeping functions, all jobs are initiated by input/output devices. We assume that all ANT nodes acknowledge the job initiations and create the entries in their respective task table locally. There is a certain degree of conceptual similarity between ANTS and FTMP in the assignment of jobs to processing units although the implementation differs in many aspects. In FTMP, as soon as the next available triad is formed, it is assigned with the next waiting job. In ANTS, the next available node grabs the next available task. The main difference being that each ANTS node maintains its own task table, which is coherent with the task tables of other ANTS nodes.

4.2 Concurrent Error Detection for Control Transfer Errors

The failure modes of an ANT node can be classified into the following two categories: *control transfer errors* and *data manipulation errors.* A control transfer error may manifest itself in the following three scenarios:

1. Fail-silent: the faulty ANT node cannot send out any message over the network due to the failure.

2. Babbling: the faulty ANT node keeps sending meaningless messages over the network and may eventually block out all communication. This is a type of fail-uncontrolled failure.

3. Tampering: the faulty ANT node is failed in such a way that the electrical characteristics of the communication network is destroyed, *i.e.* a short to the ground on the communication wire. This is a worst case scenario with a relatively small probability and can only be handled by special hardware design.

Since each computing node has an active role in seeking out work, a fail-silent node is easy to identify. A fail-silent node will not ask for a job for an extensive period of time. If a failure occurs after a new job has been acquired, then the failed node is unable to report a completion, even if it is still possible, with the fixed time period. For easy real-time scheduling, each task is defined such that it can be completed in a fixed time frame. Therefore, we may monitor the completion time of each task to detect the fail-silent nodes.

For general purpose applications, partitioning of a program into fixed completion time frame tasks requires a special compiler that can accurately estimate the execution time. This will not be the case for the proposed system. First, the system is designed to handle sonar and/or radar signal processing; the programs to be executed have been specified. Secondly, the fixed time frame constraint is not a hard constraint. The partitioning of a program must also take into account the parallelism between the tasks. In fact, a guaranteed parallelism between the tasks is the first priority in the task division. Therefore, some tasks may complete much earlier than the pre-selected time. A repeating process is required to select the proper time frame of the system.

The time limit set for the time-out checking must be greater than the time selected for the fixed time frame. The main reason for this consideration is that the communication time is a random variable. If we insist on declaring a node failure whenever it exceeds a tight time limit, we may have a great number of false alarms.

When a faulty ANT node sends out meaningless messages and jams the traffic on the communication network, it can be detected easily since all ANT nodes check all broadcasting

messages on the network. Further, each broadcasting message has a predefined format and protocol. An incorrect transmission can be detected almost immediately. The electrical characteristics of the network may also be destroyed by a failed ANT node such that further communication is impossible. This type of failures can be avoided by using a design similar to the Bus Guard (BG) in FTMP [6]. However, we caution here that this type of design is device dependent. Detailed designs can only be derived for a specific implementation. We also note that this type of failure may not be a catastrophic one. When several independent peripheral devices are used to handled buses, we may see that the probability of all bus communications being wiped out is negligibly small.

In an ANTS system, the communication of an ANT node is monitored by two other designated ANT nodes. In other words, an ANT node is responsible for monitoring two other ANT nodes. For this purpose, each ANT node in the green group is assigned with a number. Each ANT node checks the broadcasting messages constantly not only for its own message but also for the messages directed to the two ANT nodes that it monitors. When a task is acquired by an ANT node and no further message about the completion of that task was sent by that ANT node for an extensive period of time, the two ANT nodes that monitor this ANT node will sense a time out. Since all ANTS operations are asynchronous, the two ANT nodes may not detect this time-out simultaneously. The ANT node with a higher number must take the initiative and send an inquiry to the other ANT node that monitors the node in question. When a confirmation is received, the ANT node with time-out is declared faulty and is moved to the yellow group. A similar situation occurs when an ANT node sends some meaningless messages over the network.

Let's use a simple example to demonstrate this technique. Assuming that we have four nodes in the green group and they are numbered from one to four. Node 1 is monitored by node 2 and node 4, and Node 2 is monitored by node 1 and node 3, *etc.* If node 3 did not send out a message within the predefined time, and node 4 senses this situation, node 4 will first communicates with node 2 for verification. If node 2 is fault-free, it will confirm the finding of node 4 and declare that node 3 is failed.

If node 2 has failed and does not answer node 4 within the pre-defined time, node 4 will send an inquiry to node 1 to check the validity of node 2. If node 1 verifies that node 2 has not been responding for a long time, nodes 2 and 3 are both declared faulty. Note that when

nodes 2 and 3 are both faulty, node 1 will not take the initiative to send out an inquiry since it has a lower number. In this case, node 1 is waiting for node 3 to send the inquiry.

We note that the above concurrent error detection techniques for control transfer errors are for ultra-long MTTF missions. To provide higher reliability, the executions of critical tasks can be triplicated to provide the on-line error masking capability.

4.3 Concurrent Error Detection for Data Manipulation Errors

Failures may affect only the data manipulations. Computed results may be incorrect. To detect these erroneous computed results, a reasonableness checking function is associated with each job. It checks all the computed results before an acceptance. Special conditions on some operations can also be derived for the checking purpose. For example, in a Fast Fourier Transform (FFT) computation, the sum of the squared absolute values of the inputs is equal to the sum of squared absolute values of the transformed outputs. Similar checking is also possible for some matrix operations. Other type of conditions such as the *a priori* knowledge of physical limitations and special mission environment parameters will also be used in the reasonableness checking. For instance, the speed of objects monitored by a radar will lie within a physical reasonable range.

Reasonableness checking cannot guarantee full fault coverage. In contrast, a triplicated program can guarantee not only a full fault coverage, but also an immediate correction. Whereas, using the reasonableness checking, retry or re-computation may be required. However, the consideration here is that such a fault coverage may not be necessary and that the triplication degrades the system performance significantly. High fault coverage is not necessary for data manipulation errors because data integrity is not always the criterion of dependability during the entire mission. In a typical radar tracking mission, unless a close contact with enemy has been engaged, we can tolerate a few ms of not 100% correct, but reasonable, radar outputs.

If ultra-reliability is required during a particular phase of the mission, software replication as in [15, 16] can be used. This can be easily implemented on ANTS. Three entries instead of one will be generated in the task tables for each task of the critical job. This critical job is considered completed when all replications have been completed and the checking result

is successful. The voting function is implemented in software as a final task of the job, similar to that in SIFT [7]. Moreover, the concept of N-version programming [22], where the replicated programs are written by different software teams, can also be easily implemented on ANTS.

An appropriate application environment can be found in the space shuttle on-board computer system [23]. This system is designed for high availability with no fault masking capability for most of its mission time. However, during critical phases of the mission, *e.g.* take-off and re-entry, triplication with three different versions of programs are executed to ensure a high reliability in these short periods. We envision a similar application environment and thus the design philosophy is very different from that found in FTMP and SIFT.

4.4 Recovery and Re-Enlistment

When a potentially failed computing node is identified in the green group, it is placed in the yellow, checking group immediately. Of course, one of the nodes in the yellow group must be released to the green group to maintain the operational requirement. There are three different situations for different recovery strategies:

- The failed node is executing a testing function, not an operational task, when it fails. There is no need for recovery at all.
- The failed node is executing an operational task when it fails. Every fault-free ANT node will re-initialize the entry of that task such that it will again be available for execution by a free ANT node seeking work. Because the running time of each task segment must be short in order to satisfy real-time constraints, very little computations have been lost.
- The failed component is a bus, and once it is identified, will be placed in the yellow group immediately. There is no need for recovery if the bus contains no memory elements.

These recovery strategies apply to only the ANT nodes in the green group. If a node in the yellow group is found faulty, a cold start procedure is initiated and then the diagnostic

program is used to verify the checking result. When an ANT node is released from the yellow group to the green group, its task table is created from communicating with the ANT nodes that monitor it.

The system makes no distinction between a failed subsystem or a subsystem for routine check up in the yellow group. The only difference is that a failed subsystem will go through the cold start procedure before the diagnostic program checking. In addition, the failure type and the time of failure will be recorded in a failure log of the failed node after the cold start. If the result of diagnostic program checking confirms a failure, the node is removed from the system into a red stop group. If no failure could be identified, the node will be considered operational and will be released back to the green group. We call this re-enlistment. Re-enlistment occurs whenever the node failure is transient: a failure that appears only for a short time. Since a cold start is performed after a failure is identified, a transient failure will no longer exist in the subsystem.

Since intermittent and transient failures occur more frequently during run time [23], especially in modern VLSI-based systems [24], there is no need to remove a component once a failure has been recorded. The record kept at each node will indicate the number and frequency of failure history. If a computing node fails frequently, it will be removed from the system. This situation could be induced by two possibilities: the node is on the brink of a permanent failure thus increasing the frequency of transient failure occurrences; or there exists a failure which is undetectable by the diagnostic program. It is impossible to design a diagnostic program with a 100% *realistic failure* coverage, although it may guarantee a 100% coverage on modeled failures.

The re-enlistment of an ANT node with transient or intermittent failure makes the ANTS system behave similar to a repairable system in dependability evaluation. The ANT nodes in the yellow group can be considered as hot spares. As for maintainability, the diagnostic programs which are incorporated in an ANTS design can be very useful for field engineers to perform corrective maintenance. During run time checking, the purpose of the diagnostic program is to verify or to identify the existence of failures. For field repair, the diagnostic program should be augmented with the capability to locate the failures.

5 Dependability Modeling and Validation

In the following analysis, we assume that the failure rates of all components are exponentially distributed. We also deal with only the single failure cases. The main reason is because near-coincident failures or even multiple failures in an ANTS system will not induce a total system failure. Although, the inclusion of coverage factors and the use of more accurate failure function, such as Weibull, will give a more accurate evaluation [8], the presented analysis is sufficient in providing information for the system design and fine tuning. For this analysis, the field repair by maintenance engineers is not considered.

5.1 Theoretical Model

The ANTS is considered as a serial system with a KP-out-of-NP, (NP, KP), computing subsystem and a KB-out-of-NB, (NB, KB), bus subsystem. The system failure rate is the sum of the failure rates of the two subsystems [25]. Therefore, the MTTF of the system may be computed as

$$MTTF_{Sys} = \frac{1}{\frac{1}{MTTF_P} + \frac{1}{MTTF_B}} \tag{1}$$

where $MTTF_P$ is the MTTF of the (NP, KP) computing node subsystem and $MTTF_B$ is the MTTF of the (NB, KB) bus subsystem. The system is considered to have failed if less than KP processors or KB buses remain fault-free.

Even though there is no repair facility, the dependability of ANTS multicomputer system could be modeled using birth and death process due to the re-enlistment. Figure 2 shows the Markov model for the subsystems. The failure rate, λ will eventually be replaced by λ_p, the computing node failure rate, or λ_b the bus failure rate. The failures considered include both transient and permanent types. Therefore, λ_p and λ_b represent the arrival rate of transient and/or permanent failures. The birth rate in Figure 2 is ρD, where ρ is the rate for diagnostic program checking and D is defined as

$$D = \text{Prob}\{\text{failure is detectable}\} \times \text{Prob}\{\text{failure is transient}\}\ . \tag{2}$$

In other words, D is the product of failure coverage of the diagnostic program and the ratio of transient failures.

The states in Figure 2 represent the number of fault-free processors or buses. The steady-state probabilities of states in Figure 2 are expressed as [25]:

$$p_k = (\frac{\lambda}{\rho D})p_{k+1} \tag{3}$$

for $0 \leq k < N$. Since

$$p_N = 1 - \sum_{i=0}^{N-1} p_i \ , \tag{4}$$

we find

$$p_N = \frac{1}{\sum_{i=0}^{N} (\frac{\lambda}{\rho D})^i \frac{N!}{i!}} \ . \tag{5}$$

For the steady-state availability A_{ss} of a K-out-of-N subsystem, K components must be operational in an N component initial set up. The formula for A_{ss} is

$$A_{ss} = \sum_{i=K}^{N} p_i \ . \tag{6}$$

The mean time to failure (MTTF) is the expected time of system survival. Thus, the steady-state availability is the fraction of time that the system is operational [26], such that

$$A_{ss} = \frac{MTTF}{MTTF + MTTR} \ . \tag{7}$$

Here, the MTTR is the mean time to repair of the system. It is the time for a repair after the system has failed (has less than K fault-free components). Since the repair mechanism in ANTS is a diagnostic step followed by a re-enlistment step,

$$MTTR = \rho D \ . \tag{8}$$

Combining the above two equations, we find

$$MTTF = \frac{\rho D A_{ss}}{1 - A_{ss}} \ . \tag{9}$$

This model is used to evaluate the MTTF of computing subsystem and that of bus subsystem: $MTTF_P$ and $MTTF_B$, respectively. The MTTF of the system is then computed using equation (1).

5.2 Mean Time To Failure

Let us consider a design goal of the ANTS multicomputer system achieving a MTTF of more than 20 years, or more than 175,200 hours. We use the model developed in the previous section to examine the effects of computing node failure rate and the D to the system MTTF. We show that the ANTS concept could indeed tolerate a wide range of computing node failure rate as well as D (product of diagnostic program failure coverage and ratio of transient failures).

5.2.1 Effects of Computing Node Failure Rates

Figures 3 and 4 show the system MTTF of various ANTS configurations and a conventional hybrid N-modular redundancy system (HNMR). The graphs are for (NP, KP)=(8, 3), (8, 4), (16, 10) and (16, 12), respectively. For the HNMR system, we consider a 3-out-of-8 system with 32 standby spares. We further assume that these 32 cold spares will not fail while standby. According to [25], the MTTF of a HNMR(N, M)/S system is derived as:

$$MTTF_{HNMR} = \frac{S}{N\lambda} + \sum_{i=M}^{N} \frac{1}{i\lambda} \tag{10}$$

where λ is the component failure rate.

Figure 3 assumes a bus failure rate of 10^{-5} per hour, while Figure 4 assumes that the bus failure rate is 10^{-4} per hour. In each case of ANTS configuration, decreases in MTTF numbers are observed when the bus failure rate is higher. However, the decrease in MTTF is insignificant compare to the 10 times increase in the bus failure rate. In the above results, we assume ρ=0.1, or an averaging diagnostic program checking time of 10 hours. This is indeed a worst case assumption. Further, D=0.8 is assumed.

Obviously ANTS yields a much higher MTTF than a conventional HNMR system. From Figure 4, an ANTS system with NP=16 and KP=12 could achieve more than 20 years of operations even when the computing node failure rate is 3×10^{-4} per hours. The above results strongly imply that there is no need for ultra-dependable computing nodes. A relatively inexpensive implementation of ANTS to achieve the MTTF goal is feasible.

5.2.2 Effects of Failure Coverage and Transient Failure Ratio

One of the reasons that the ANTS concept is a significant improvement with respect to the conventional HNMR is the fact that the failed computing nodes in ANTS are re-enlisted after a successful diagnostic program check out. The dependability of a HNMR could be enhanced using the same technique, but the system computational performance of HNMR is not comparable to that of ANTS. For instance, the HNMR(32, 8)/32 system has a total of 40 computing nodes: eight active ones and 32 cold spares. The entire system is used as a uniprocessor system. Whereas an ANTS system is a true distributed computer system, all 40 computing nodes could be fully utilized to perform useful tasks.

According to [25], a high system availability could be achieved by one very efficient "repairman" rather than by an unlimited number of repairmen. In the above dependability model, we use ρD to model the ANTS "repair" process. In Figures 5 and 6, we plot the system MTTF against the values of D to determine the effect of low diagnostic program failure coverage and low transient failure ratio.

The plots in Figures 5 and 6 have identical parameters, except for the bus configurations. In Figure 5, the best case configuration is a (16, 8) ANTS system, which achieves the desired goal with D as low as 0.35. In the worst case, a (32, 28) ANTS system, the lowest possible D is about 0.75. From Figure 6, we find that the requirement of D decreases with a (4, 2) bus configuration for (8, 4) and (32, 28) configurations. The (8, 3) and (16, 8) configurations need a higher D to achieve the same level of MTTF when using a (4, 2) bus configuration.

We assume here that ρ=0.1, or a 10 hours diagnostic program execution time. We emphasize that this is a worst case assumption. The actual diagnostic program should execute at a much faster rate.

There are two parameters involved in deriving D: diagnostic program failure coverage and transient failure ratio. The transient failure ratio is an environmental parameter, in which we have no control. The failure coverage is the percentage of failures that can be detected by the diagnostic program. A higher failure coverage is possible by a carefully designed diagnostic program. We note that there is a trade-off between the failure coverage and the program execution rate. Intuitively, a diagnostic program with higher failure coverage takes longer time to complete.

Assuming that the failure coverage of the diagnostic program is 90%, to reach D=0.35 means that the transient failure ratio must be 0.39. For D=0.75, the transient failure rate must be 0.83. In other words, given a transient failure ratio, there exists an ANTS configuration that achieves the goal that MTTF exceeds 20 years.

The study on space shuttle computers [23] assumes that the "self-test" program, which is equivalent to the diagnostic program in ANTS, has a fault coverage of 96%. Also, a 2:1 and a 4:1 "transient-to-solid", *i.e.* transient-to-permanent type failures, are assumed in the dependability analysis. Using the terminology in this paper, a 2:1 ratio means the transient failure ratio of 0.66, and a 4:1 ratio means the ratio is 0.8.

5.3 Fault Tolerant Features Verifications

5.3.1 Fault Injection Experiements

The concurrent error detecting mechanism and other integrity maintenance protocols were tested using simulated faults. The tests included cutting off the power supply to nodes or to parts of the hardware, intentional triggering of faults by transmission of invalid data over the bus and simulation of memory parity errors. Isolated and multiple faults were simulated to fully test the correctness of the monitoring scheme. The observed results conformed to the specified requirements in that each fault free node reaches an accurate diagnosis of the fault conditions of the remaining nodes, without any restriction being placed on the number of faulty nodes or fault patterns.

The fault tolerant mechanism always correctly reconfigured the network in all the test cases. Under extreme load conditions, it took the ANTS system (with 9 nodes) a maximum time of one minute to reconfigure itself to a stable state. Under nominal conditions of load and faults the mean reconfiguration time was about 10 seconds. The reconfiguration time is the time interval between the occurring of a fault and the time the network becomes fully operational again, *i.e.*, the faulty nodes have been moved to the yellow group and available ANT nodes at the yellow group are re-enlisted to the green group. The reconfiguration time is related to the number of faults occurring at the same time and to the communication network latency.

Tables 1 and 2 show the typical reconfiguration times observed for the *isolated faults* and

chained faults cases, respectively. The rather irregular variations in the reconfiguration time is due to the fact that, the monitoring protocols share the communication bus with the rest of the system. In such a situation the reconfiguration time is affected to a large extent by the application tasks that are running. The number of simultaneous or near simultaneous faults were limited to 6 in all the test cases. With more than 6 simultaneous faults and at high load levels the communication bottleneck was found to cause the system to become unpredictable.

5.3.2 ANTS Performance with Built-In Fault Tolerant Features

In order to study the impact of the inclusion of aforementioned fault-tolerant features to the overall system performance, we perform the following experiment. First, the speed up factors of ANTS without fault-tolerant features are measured. The fault-tolerant features of the ANTS kernel can be turned off by a selected option. Then the speed up factors of ANTS with fault-tolerant features turned on are measured. The test run results are shown in Figure 7. Figure 7 clearly shows that one of the major design goals has been achieved. The ANTS is designed to exploit the inherent fault tolerant features of the distributed system. We expect that a successful implementation of ANTS should have a minimum impact on the system performance. Test runs yielded a figure of 5% for the performance degradation due to the addition of fault tolerance. We believe that this low figure is a strong evidence to support that the inherent fault tolerance has been well utilized.

We have shown previously in [1] that an ANTS system can achieve an extremely long MTTF. Part of this MTTF improvement is due to the re-enlistment of ANT nodes with transient type failures. We note that the ratio of transient faults to all faults depends to a large extent on environmental conditions. In a documented case [14], the transient faults account for 20% to 33% in low earth orbit conditions. This clearly brings out the effectiveness of the recovery process of the ANTS fault tolerant mechanism. The penalty paid for obtaining such an improvement in MTTF is a minimal trade-off in performance, as shown in Fig 10. The loss of performance is a function of the number of nodes in the system and the mean performance degradation with 4 and 8 nodes was 2% and 5%, respectively. This is relatively small compared to the magnitude by which the recovery and re-enlistment

process extends the useful life of the system.

6 ANTS Kernel Version 1

The first implementation of ANTS is carried out in a configuration as shown in Figure 8. The computing nodes in this case are Sun workstations, and the communication bus is an Ethernet. Of course, ANTS can be implemented on any type of parallel processing configurations. The choice of this particular system organization is due to the availability of hardware facilities.

The ANTS kernel is built on top of the SUN OS and the TCP/IP [27] communication protocol. A detailed view is shown in Figure 9. There are four major modules in the ANTS kernel: System Manager (ANTS-SM), Fault Tolerance Monitor (AFDRP), Task Scheduler, and Inter Node Communication Control. The functions of these major modules are presented in the following, except for the AFDRP which will be presented in the next section.

6.1 ANTS System Manager

The ANTS system manager or ANTS-SM maintains a record called system information block (SIB). The SIB consists of three major categories of records as shown in Figure 10. These records are updated either due to interrupt-driven events, such as the system time, or due to messages oriented from other nodes, such as task completion record.

6.1.1 Synchronization and the System Clock

In ANTS there is no need for highly synchronized clock mechanisms. The ANTS nodes run in loose synchronization, the only need for a synchronized clock in ANTS arises when time-stamps have to be generated. The ANTS Task scheduler is designed such that even for the time-stamps a high degree of synchronization is not required.

Based on these requirements the ANT-SM maintains a simple clock that counts in milliseconds. The clock is initialized during start up and is periodically checked at long intervals with neighboring nodes to trap possible erroneous values (this is done as part of the concurrent error detection mechanism at the kernel level). Other than this the precision of the

system clocks is such that it does not need to be adjusted for periods far exceeding the system uptime.

6.1.2 ANTS Address Resolution and Initialization Protocol

This protocol is concerned with the process of initiating the entrance of a node into the *green group*. First the node that intends to enter the network randomly picks a slot from the list of valid ANTS addresses, it then broadcasts a query with the chosen ANTS address as part of the message. There are two possible outcomes of the query, first the requested ANTS slot may be vacant in which case one of the nodes in the network, *i.e.*, the one with the next higher ANTS address assigns itself the task of initiating the incoming node into the network. The task of initialization includes the transfer of the current task table and the System Information Base. Also if the fault tolerance feature is enabled the neighboring monitoring nodes reassign themselves to monitor the new node. The monitoring mechanism in the new node becomes active and starts monitoring its adjacent nodes.

If the selected ANTS address posted in the query clashes with that of an ANTS node that is already in the network then that node by default becomes the initiator. This node refers to its SIB and picks a valid ANTS address that is not in use and assigns it to the incoming node. Also, as in the previous case the initiating node transfers all the necessary information to the new node.

6.1.3 Debugging and System Diagnosis Features

The ANT-SM also provides a run time window display of the functioning of the system. Some of the data that is displayed includes the first 10 tasks on the task list, the currently executing task, the status of INC and the list of active nodes in the system. The display basically provides continuous status information of the running system besides showing some cumulative data such as the total traffic over the communication network and number of tasks executed.

The other necessary system functions in the prototype are handled by the host operating system the ANTS kernel is running on. In future versions it would be advantageous to completely eliminate the host operating system interface to the hardware and implement

the required features taking into account the requirements of the ANTS system.

6.2 Task Scheduler

The ANTS Task Scheduler distributes the tasks waiting to be executed among all the nodes in the operational pool of computing nodes. The most important feature of the algorithm is the maintenance of a common coherent task list at all nodes.

6.2.1 Maintaining Coherency of Task Lists

Each node in the system has a data structure which reflects the state of all the tasks in the system and the list is assumed to be identical at all nodes. This data structure is comprised of an ordered list of tasks that need to be executed. Tasks are added to this list whenever they are invoked and are deleted when they have been taken for execution. Maintaining the coherency of the task lists is done by passing messages among nodes. Three basic operations are used to establish the task list coherency.

1. *Append*: Whenever a task originates, an *APPEND* message is broadcast over the network. Along with the message, a data record containing all the information about the task that is being added is also broadcast. Each node that receives the message adds the data record to its respective task list.

2. *Delete*: Any time a node picks up a task for execution, it send out a *DELETE* message along with a data record identifying the task that is being taken up for execution. As before, every node on receiving this message deletes the task from its own task list.

3. *Add First*: This operation is similar to the APPEND, except that the task that is to be added is put on top of the task list instead of appending to the end. This is useful for inserting a high priority task and can be directed to a particular computing node.

6.2.2 Selection Policy

There are two selection policies implemented in the Task Scheduler. One type of selection policy is for applications that need to be executed on a first come first serve basis. In this case, a *Simple Selection Policy* is used. The next task to be executed is the one that is on

top of the task list. So, whenever a node is free it grabs the top most task from the task list if it is available for execution.

For applications that are data intensive, the execution of successive tasks with direct predecessor-successor relationship on different nodes require that huge volumes of data be transferred between the nodes that execute them. The communication cost involved could be reduced to a great extent in such cases by executing these tasks on the same node. The data that the tasks need are already stored locally and practically no inter-node communication is required.

When a node is free to select a task from the task list, instead of just selecting the task at the top of the list, a *look ahead* into the list is performed. Now, if another task which is a direct successor of the just completed task is found, it is chosen for execution instead of the one on top of the task list. This would result in considerable saving in terms of the data traffic between nodes. The depth of look ahead will be determined by factors such as the overhead involved, task queue size and other application specific issues. Note that a simple selection policy is obtained when the look ahead depth is one.

6.2.3 Contention Resolution

The distributed task scheduler at each computing node selects the next task to be executed from its own copy of task list. Because of the communication delay, it is possible that several computing nodes may select the same task to be executed almost simultaneously. If such scenario occurs, the task scheduler will receive messages from other computing nodes stating the the task has been selected for execution at other nodes. As shown in Figure 11, the time-stamps indicating the time the task is selected are compared. The computing node with a higher number in the times-tamp will abandon the execution and will proceed to select another task from the task list.

This simple mechanism has been shown to be effective during experimental runs. It is also quite efficient since only very little overhead in comparing the time-stamps is needed.

6.3 Inter Node Communication Control

The ANTS kernel Inter Node Communication facilities are built on top of the TCP Internet Protocols [27] which is available in the Sun OS. The purpose of adding this kernel on top of the available protocol is to customize the communication services available to best suit the needs of the ANTS system.

All the messages handled by the INC conform to the predefined formats. Any message that does not follow the known patterns results in the node that sourced the message being put under test and diagnosis. The first byte of all messages indicates the type of message it carries. Figure 12 shows the bit patterns of the first byte and the message it identifies. Figures 13 and 14 show the message format for task scheduling and data transfer and for AFDRP, respectively.

A salient feature of the network interface is that it's network number of address is fully dynamic, in that the address can be modified by the ANTS Integrity Maintenance Protocol at run-time. The network number consists of a 16-bit integer number which leads to a maximum possible number of 65,536 nodes. A main reason for this arrangement of dynamic ANTS network number is so that no critical region exists by pointing to a fixed number as in a common local area network. The dynamic assignment of a network number enables us to freely remove or insert a computing node without seriously interfering with the system performance.

6.4 Fault Tolerant Features

The fault tolerant features reported here are those implemented at the ANTS kernel level for the experiments. Data manipulation errors are not detected by the fault tolerant techniques described here. They are detected by concurrent error detecting techniques to be implemented at the task level. For instance, in a high availability application, reasonableness checking on the computed results can be easily imposed. For high reliability applications, duplicated or triplicated tasks can be used.

6.4.1 Run Time Partitioning

During run time ANT nodes are partitioned into three groups: the up-and-running *green* group, the stop-and-checking *yellow* group and the *red* group for discarded faulty nodes. The relations between run time groups are illustrated in Figure 15. The number of computing nodes allocated for the green group must always be greater than the minimum requirement at any given time. The rest of the computing nodes are then in the yellow group unless they have been found to fail, in which case they are removed to the red group. The grouping is based on the status of the computing nodes. Therefore, there is no visible physical grouping, and no need for special hardware design for switching between the two run-time groups.

There are two ways that an ANT node may be moved from the green group to the yellow group: (1) when it is time for a routine self diagnosis, or (2) when a node has been identified as faulty. Since each ANT node seeks its own work for itself, placing an ANT node in the yellow group is accomplished by making the diagnostic program the highest priority task for it through a timer. An ANT node may be moved from the yellow group to the green group if the diagnosis result proves fault-free.

All faulty nodes are placed in the red group. When no manual repair is available, the nodes will be indefinitely remained in the red group. A node in the red group can be moved back to the green group only if it has been repaired.

6.4.2 Concurrent Error Detection

Since each ANT node has an active role in seeking out work, malicious failures, such as fail-silent or fail-uncontrolled [15], are easy to identify. A fail-silent or fail-uncontrolled ANT node will not ask for a job for an extensive period of time. The concurrent error detection at the ANTS kernel level is based on *bus eavesdropping*. This requires a communication interconnection system where broadcast is cheap. Ethernet based networks are ideal examples of such an interconnection medium.

If a failure occurs after a new job has been acquired, then the ANT node will not report a completion of the task within a reasonable time. For easy real-time scheduling as well as concurrent error detection, we partition jobs into tasks such that each task can be completed within a fixed time frame. Therefore, time-out detection can be used as the first layer of

concurrent error detection mechanism.

At the ANTS kernel level, we implement a monitor scheme, ANTS Fault detection and Recovery Protocol (AFDRP), that is similar to that proposed in [28]. Each ANT node monitors its two neighboring nodes. Conversely, each ANT node is monitored by its two neighboring nodes. All ANT nodes are assigned with node addresses. These addresses are used at run time to determine the neighboring nodes.

For the case shown in Figure 16, let us consider node M. Each time M receives a message from H or L indicating that they are taking up a task for execution M updates its *Last Activity* record for that node. This forms one phase of the protocol. Each time node M finishes executing a task it enters the *Verification* cycle, as shown in Figure 17. Every node in the system runs a verification cycle to verify the correct operation of its upstream neighbor (H in this example). The verification cycle begins by checking if the node under test has been silent for more than a predefined timeout factor. If this is not the case then node H passes the test and M exits the verification cycle. If node M finds that node H has remained silent for more than the timeout factor, it proceeds with the *Confirmation Cycle.*

The confirmation cycle begins with node M communicating with the other node monitoring H (which is H2 in this example), requesting a verification of the activity of node H. When H2 receives this request it looks up its database and sends back a reply to node M. The reply consists of a *Concur* or *Differ* message depending on the activity report of node H at H2. On receiving a *Concur* reply from node H2, M enters the *Recover* and *Restart* (R&R) cycle, as shown in Figure 18. The confirmation cycle effectively prevents faulty nodes from removing operational nodes from the network.

Beside the above protocol, other checking functions such as periodic self-diagnosis, memory parity checks, cyclic redundancy checks on inter node messages and hardware integrity checking can also be selectively done depending on the available resources. Detection of any faults during these checks also triggers the recovery and restart cycle.

6.4.3 Recovery and Re-Enlistment

Once the concurrent error detection mechanism successfully detects a fault, it triggers the recovery and restart (R&R) cycle. The first step in the R&R cycle consists of M, as in

previous example shown in Figure 16, broadcasting a *Remove From Net* (RFN) message for node H. ANT nodes that receive this message enter different recovery modes based on their logical position (ANTS Address) in the network. If node H is partially functioning and receives this message it immediately enters a full diagnosis cycle, where it undergoes a thorough test of itself. If the RFN message is received by a node that is monitoring the node that is being removed, the node resets itself to monitor its new neighbor.

All the ANT nodes modify their *System Information Block* (SIB) to reflect the removal of the node in question. The next step in the R&R step consists of re-enlistment, this step is carried out only if a functional ANT node is available from the yellow group. If an ANT node S is available, node M brings it on-line by first allotting it the address of the removed node and proceeds to initialize the new node by providing it with the current Task List and other system parameters. Once on-line, node S starts normal execution which includes the monitoring of its adjacent neighbors. While providing the new node with the Task List, node M inserts the task that failed to complete on node H on top of the list, this effectively results in the task being restarted on node S. This marks the completion of the *Recover and Restart* cycle and the system returns to a normal state. If an additional ANT node is not available, node M skips the re-enlistment cycle and simply restarts the aborted task itself.

Since intermittent and transient failures occur more frequently during run-time [23], especially in modern VLSI-based systems [24], there is no need to remove a component once a failure has been recorded. The record keeps at each ANT node will indicate the number and frequency of failure history. If an ANT node fails frequently, it will be removed from the system to the red group. This situation may be induced by either of the following scenarios: the node is on the brink of a permanent failure and thus the increasing frequency of transient failure occurrences; or the failure is undetectable by the diagnostic program. It is impossible to design a diagnostic program with a 100% *realistic failure* coverage, although it may guarantee a 100% coverage on modeled failures.

6.4.4 Error Handling Capability

The monitoring scheme discussed above not only detect single isolated faulty nodes but multiple faulty nodes as well. In the following we shall use examples to demonstrate the

detection of multiple node failures.

For the isolated node failures shown in Figure 19, the AFDRP has no trouble in detecting them. For each failed node, the two nodes that perform the monitoring are both fault-free. The communication can be established for the verification cycle and proceed directly to the R&R cycle.

Consider the multiple (chained) node failures case shown in Figure 19 where nodes 2 and 3 are faulty. Node 1 discovers the fault in node 2 during the *Verification* cycle and requests confirmation from node 3. While requesting confirmation from node 3, node 1 starts a timer and if it does not receive a reply from node 3 before the timer expires it proceeds to run a verification of node 3 by requesting a status report of node 3 from node 4 and the cycle continues.

Based on the assumptions made previously, we can see that in this example node 1 will not receive a valid reply within the timeout period and node 4 will concur with node 1 about the failure of node 3. Now node 1 begins the R&R cycle. This verification process can be extended to handle as many *Chained* or *Multiple Consecutive Faults* as necessary. By extending the above two cases it is seen that any combination of the above two cases will be successfully handled by the protocol. It is also seen that this combination covers all possible fault patterns.

The main advantage of the above scheme is that the monitoring of nodes and the fault detection process add no additional traffic over the communication network. The protocols function by monitoring other system communication messages and hence fault tolerance is achieved at the least communication expense.

6.5 Experimental Results

The ANTS prototype was tested on Sun Sparc workstations running Sun OS. The ANTS communication mechanism was layered on top of the Internet Protocols and the intercommunication network used was an Ethernet LAN running at 10 Mbits per second. The ANTS code was written in C++ and compiled using the GNU C++ compiler. The test application program was emulated using timing loops and an inter task data transfer size of 1.5KB was assumed. Predecessor and successor relationships of tasks are either generated randomly or

are assumed to resemble that of the FFT computation. In addition, some randomly generated tasks with randomly generated data dependency relationships are also used. Due to the limitation in the number of available nodes, the test runs were limited to a maximum of nine nodes. Tests were done for three different *Task Slice Time* factors and a comparison of performance with and without fault tolerance was also made to study the performance trade off implications of using the ANTS Integrity Maintenance Protocols.

6.5.1 ANTS Performance without Fault Tolerant Features

Figure 20 shows the performance figures obtained from the test results. During these test runs, the AFDRP or the Fault Tolerance Monitor is disabled. The modulation in programming ensures that these results are obtained with no interference from the disabled modules. The following inferences can be drawn from the results:

First, as is to be expected the speed up factor is not linear. The speedup obtained gradually decreases with the addition of more nodes. The decrease in speed up is more pronounced with an increase in the number of nodes beyond 6. This effect is due to the traffic bottleneck imposed by the single communication channel. Addition of more communication channels or an increase in bandwidth of the communication media is necessary to achieve a better performance with more nodes in the system.

Another noticeable aspect of the performance figures is the higher linearity in speed up with higher Time Slice factors, *i.e.*, task execution time. This is evidently due to the fact that less overhead is involved with larger task fragments. The time slice factor will however have to be limited to a reasonable value to facilitate faster trapping of faulty nodes. If the time slice factor were to be too large, a faulty node could unrecoverably damage the cooperative processing environment of the ANT system.

A third prominent feature is that with a large number of nodes in the system the speed up tapers off to an almost constant value. This value is a function of the total available bandwidth of the communication medium. This compares well with other distributed computing systems where beyond a certain point the speed up curve inverts and results in worsening performance [29]. This is due to the fact that with more nodes there are more messages that contend for the communication medium. In ANTS the communication traffic is not

adversely affected by the number of nodes in the system.

We also observe that the average communication delay is about 200 to 250ms. There is a strong relationship between the average task execution time, the average communication delay, and the speed up factor. Of course, when a high speed network is available, a shorter task execution time can be used.

6.5.2 Effects of Communication Delays

As in any distributed computing environment, communication is a major part of the ANTS operations. Broadcasting of messages to all ANT nodes is necessary when taking a task or completing a task. The performance of an ANTS system may be directly affected by the underlying communication delays. In this experiment, we use a 10Mb/s Ethernet local area network. In Figure 20, we see some of the effects of the communication delays as it relates to the average task execution time.

In Figure 21, the plots show the speed up curves as we intentionally extend the average task execution time to as high as five seconds. The purpose is to compromise the effect of the communication delays and to have a preview of what may happen with a higher speed local area network. By increasing the average task execution time, the broadcasting needs spread out more evenly in time. Communication contentions become unlikely. We see in this case that the speed up curves are closer to a linear speed up with a long task execution time. This means that changing to a high speed local area network will have positive effect on the ANTS performance.

7 ANTS Kernel Version 2

In the version 2 implementation of ANTS kernel, we concentrate on the performance issues. First, the new ANTS kernel will be resided in a workstation as a UNIX daemon. This means that the ANTS kernel will consume virtually no computing resources when it is idle. In addition, several new features are added to the version 2 which enables a smooth handling of communication messages. The fault-tolerant features remains the same as in version 1 and thus will not be discussed in this section.

7.1 Conflict and Coherency

Since each computing node in an ANTS distributed computing environment is actively seeking new tasks from its own task queue, conflict may occur in which two or more ANTS nodes pick up the same task at roughly the same time. In version 1, any conflict is resolved by comparing the time-stamps of all ANTS nodes who have picked the same task. In version 2, the same mechanism is still in use. However, a new policy is installed so that the frequency of conflicts in the system may be reduced significantly.

The frequency of conflicts can be greatly reduced by the following scheme. When a new job is initiated, not all computing nodes receive the tasks in the same order. For example, consider a new job with tasks 1 to 12 as shown in Figure 22. When this new job is initiated into local task queues of computing nodes, some may receive tasks in a sequence of 1, 2, 3, 4, 5, 6, 7, 8, etc., while others may receive tasks in a sequence of 3, 4, 1, 2, 7, 8, 5, 6, etc. We note that tasks 1 and 2 and tasks 3 and 4 are completely independent. By shuffling their orders in the sequence of tasks will not change the computational outcome, but will greatly reduce the possibility of a conflict. In the above example, half of the computing nodes may compete for task 1 while the other half compete for task 3. Of course the possibility of a conflict can be further reduced by a further diversity in tasks sequencing.

The coherency of local task queues cannot be guaranteed in an ANTS system, because the updating messages arrive at their destinations with with arbitrary communication delays. There are several factors that constitute a communication delay, namely: the communication medium, the protocol, the possibility of an on-going communication, the local peripheral response delay, etc. We may say that although broadcasting is used to announce the ownership of a task, not all computing nodes receive the message and update their queue exactly at the same time. However, we realized that this may not be a problem at all for the ANTS operations. The reason is that such inconsistency in local task queue contents does not significantly affect the performance or the dependability of the system.

7.2 Communication Delays

Communication delay is an important factor for any parallel and distributed computing system. In the experimental ANTS system, we are limited by the speed of the 10Mb/s Ethernet

LAN. Consequently, we can consider only the applications with coarse grain parallelism. Of course, this is the same limitation for all similar systems mentioned in Section 2.

Since the raw message passing speed is governed solely by the underlying hardware equipment, the only thing we can do via software is to reduce the frequency of communication. Besides job initiations, typical communications between ANTS nodes are:

1. Ownership message: A node broadcasts message to claim the ownership of a task. This happens immediately after an idle node picks up a new task to be executed.

2. Completion message: Upon the completion of the local task, a node broadcasts a message to inform the other nodes about the completion.

3. Data request message: When a task has been acquired by a node and part of the data needed by this task is not available locally, a node broadcasts a message to request the data.

4. Data transfer message: A node sends out data in reply to the data request message. Depending upon the size of the data, this message can be packed with the Completion message.

5. Add Task message: All nodes are empowered to add tasks, anytime, to an already existing task list. The request of a node to spawn new tasks is a task in itself. In that case, the concerned node broadcasts a message to all other nodes to update their local task queues, by appending these newly spawned tasks to their task list.

In order to reduce the frequency of communications, we have established the following conventions:

1. When a node picks up a new task, this node is either previously idle due to the lack of tasks in the local task queue, or has just completed a task. The former situation usually occurs only during initiation or during a shortage of tasks to be executed. The latter cases occur more often. In other words, we may say that a node usually picks up a new task when it finishes an old task. Therefore, we combine the ownership message with completion message in one broadcast effort.

This policy also helps reinforce a previous policy where a node will try to pick a new task for which the data requirement can be satisfied locally. When a task is completed, no other node is aware of this fact except the local node until the local node picks up a new task and broadcasts the message. The possibility of conflict in claiming ownership of a new task is also reduced.

2. The data request message can also be combined with an ownership and completion message. The only time a node requests outside data is when the new task calls for data that is not available locally. Obviously this occurs only when a new task is picked. If the size of the required data is not large, the whole data set can be combined with the ownership and completion message. But, if the data size is huge (say, all the elements of a very large matrix), broadcasting the data to all nodes would turn out to be wasteful. In such a scenario, the data can be made available upon request.

3. The new task message due to the spawning of new tasks from an old task can also be combined with the above message. There is a minimum performance loss since some task may request spawning of new tasks in the middle of its execution. Here, we postpone that action until the completion of the old task.

Therefore, effectively, only three types of communication are required: *job initiation* (JI), *ownership, completion, data request, add task* (OCDA), and *data transfer* (DT). The JI messages is a one-time effort and occur only during initiation of a new task. The number of needed OCDA messages for a job is about the same as the number of tasks in that job. The frequency of DT depends on the data dependency of the algorithm. In fact, a carefully designed ANTS-executable algorithm should take this into consideration for high-performance computation.

7.3 The Implementation

The ANTS prototype was tested on Sun Sparc workstations running Solaris. The ANTS communication mechanism was layered on top of the Internet Protocols and the intercommunication network used was an Ethernet LAN running at 10 Mbits per second. The ANTS code was written in C++ and compiled and tested using, both GNU G++ and Sun CC

compilers. In the earlier prototype, the test application program was emulated using timing loops and an inter task data transfer size of 1.5KB was assumed. Predecessor and successor relationships of tasks were either generated randomly or were assumed to resemble that of the FFT computation. In addition, some randomly generated tasks with randomly generated data dependency relationships were also used.

Such assumptions are now unnecessary. At the time of writing this paper, ANTS has been fully tested to run the L U Decomposition algorithm, Matrix multiplication, Replica-Subspace Weighted Projections (RSWP) algorithm [30] and the Multiple-Uncertainty Replica-Subspace Weighted Projections (MU-RSWP) algorithm [31]. The RSWP and the MU-RSWP algorithms perform source localization in an acoustic waveguide. These algorithms, written in Matlab M-script, are run within ANTS by initiating a Matlab process (engine) using the External Interface Library of Matlab. Also, there is no restriction on the size of data being passed across the network. If the data size exceeds 64KB (the maximum limit of the an ethernet packet), the data is broken into an appropriate number of packets. The Matrix multiply example, was used primarily to test this capability of the ANTS system.

Due to the limitation in the number of available nodes, the test runs were limited to a maximum of nine nodes. Tests were done for three different *Task Slice Time* factors and a comparison of performance with and without fault tolerance was also made to study the performance trade off implications of using the ANTS Integrity Maintenance Protocols.

8 Designing Algorithms for ANTS

The ANTS concept provides a convenient way to perform distributed computing. The kernel or the distributed operating system itself already envisions several performance improvement features. Further improvement on performance is also likely by algorithms that are designed specifically with the ANTS operating principal in mind. In particular, granularity and inter-task communication frequency are the two major factors to be considered.

8.1 C Simulator

A simulator was written in the C programming language to run on a PC or on workstations. It is being used to test various strategies for the implementation of subroutines for computationally intensive algorithms. The simulator permits the inclusion of timing estimates of data input, internode communication, task time, and task precedence in terms of tables. Several versions were used to try several ways of having nodes select tasks. The most recent examines predecessors and estimates the communication time. A further enhancement being considered is to let the test look ahead as well as back to reduce internode communication. Doing this manually showed some improvements in the FFT algorithms.

8.2 Computational Algorithms

We are considering algorithms used in large cpu-intensive applications. We have programmed and tested several subroutines for FFT's, QR algorithms, and L U decomposition. These dominate large applications such as finite element methods, large-scale fluid flow problems, and digital signal processing. The names and functions of our subroutines follow those used in Matlab[2]. This will permit their use in Matlab Mex files and make them accessible in Matlab programs. These matrix and Fourier algorithms may have their greatest impact on multiprocessor machines, for which the communication delays will be much smaller than on a network of workstations.

8.2.1 FFT

Several ways of decomposing the radix 2 FFT algorithms were designed and run on the simulator. One of them is being implemented on a network of workstations. The algorithm is being generalized for powers of primes. Subroutines for powers of primes can be used for Fourier transforms of multiple arrays and as parts of more general mutually prime factor programs for arbitrary transform size N.

[2]Matlab is a registered trademark of Mathworks Inc.

8.2.2 Linear Equation Solvers

The basic algorithm of many linear equation programs is the L U decomposition. Given the problem of solving $A \cdot x = y$ for x, one computes lower and upper triangular matrices L and U, respectively, such that $A = L \cdot U$. Since this requires n^3 operations, the task grows rapidly with problem size. The L and U may be used repeatedly in solving for many y's. They also give the matrix inverse and the determinant. The L and U are also very useful in some methods for very large eigenvalue computations.

8.2.3 QR Algorithms

Given a rectangular $m \times n$ matrix A with $m \geq n$, this algorithm uses Householder elimination to determine an $m \times m$ matrix Q and an $m \times n$ matrix R such that $A = Q \cdot R$ and the elements of R satisfy $r_{ij} = 0$ for $i \leq j$. This can be used to solve linear equations for a best least squares solution where there are more equations than unknowns. The elimination method will also be used in the eigenvalue subroutines.

8.2.4 Poisson Equation Solver

We wrote and tested a C++ program for solving a Poisson equation, *i.e.*, a 2-dimensional second order elliptic partial differential equation with zero boundary conditions for the solution and for the right hand side. A Fourier transform method was used. For this, Fourier transform subroutines for doing row and column FFT's of 2 dimensional arrays were developed.

This is a good test case for ANTS since this calculation is done on very large arrays in numerical weather prediction, the calculation of magnetic and electric fields, and large diffusion problems. The parallelism is quite simple. The program can do simultaneous FFT's on sets of rows on different processors followed by a similar treatment of columns.

8.3 RSWP and MU-RSWP Algorithms

Matched-field processing is an acoustic signal processing technique used to locate signal sources by matching the received signal to a model. The models are full-field acoustic propagation models in that they include the parameters of the ocean environment. However, these

methods can be sensitive to imprecise knowledge of the ocean environment parameters. The replica-subspace weighted projections algorithm (RSWP) [30] and the multiple-uncertainty replica-subspace weighted projections algorithm (MU-RSWP) [31] are matched-field processing techniques which are robust to environmental parameter uncertainty. Prior to the processing of data, these techniques require the computation of signal models. These models are in the form of vectors with one set of vectors for each signal frequency. Typically, this is a computationally intensive task requiring the generation of tens-of-thousands of vectors per set.

The program has been written in Matlab and tested. Hence, first, we develop an interface between Matlab and ANTS. Matlab is a script language from MathWork Inc. and has been used in this Department for Digital Signal Processing Teaching and Research purposes.

8.3.1 Matlab Interface

In order to execute a segment of a Matlab program without invoking interactive Matlab environment, we use the off-line Matlab engine. A Matlab engine can be initiated by the ANTS kernel as a child process. The ANTS kernel then feeds the Matlab engine with the segment of Matlab program to be executed line by line.

During the implementation, we realized that when the Matlab engine was killed after the completion of the segment of Matlab program, all data that it has generated are wipe out from the main memory. We first tried a method of storing the data to hard disk in a temporary file before the Matlab engine session ends. However, in that case, the data read and write time to/from hard disk become a significantly factor in performance. We then decided to use a UNIX system function to force the sharing of the main memory space allocating to the Matlab engine with ANTS kernel. Since Matlab engine is considered as the child process of the ANTS kernel in the view of the UNIX operating system, this is easily accomplished. In essence, the ANTS kernel will retain the data generated and kill the Matlab engine as it completes the assigned task.

8.3.2 Task Execution Times

The RSWP algorithm has 12 tasks where tasks 2 to 12 can be executed in parallel and are all dependent on task 1. Table 3 lists the individual execution times when running on a Sun SPARC5 workstation. Note that these task execution times spread in a wide range from 5.2 seconds to 51.7 seconds. Of course, tasks 6 to 12 have approximately the same execution time. Even though the task execution times vary, no load balancing problem arose during experiments. The ANTS kernel seems to be able to adapt quickly to this type of situation without difficulty.

8.3.3 Experiment Results

Figure 23 shows the speed up factors in experimental runs. Up to 10 Sun SPARC5 workstations were used to run the program. The speed up curve is near linear when the number of nodes is no more than 3. Then the speed up factor is slow to increase compared to the number of ANTS nodes. Nonetheless, we still see a speed up of more than 6 when 10 ANTS nodes participated in the computation.

There are several factors that prevent a linear speed-up. Besides communication delay and other typical performance overhead, in this case, we also have the overhead of indirect execution. The real computation is done by the Matlab engine which is initiated by the ANTS kernel. We believe that a much better speed-up can be obtained if the Matlab engine can be modified to run ANTS tasks directly. Conversely, if we can run Matlab program directly using ANTS kernel, then the enhancement in the speed up factor can be expected.

9 Conclusions

We have presented in this report the dependability analysis of the ANTS ultra-dependable multicomputer system We have shown that the concept of run-time partitioning for diagnostic program checking and the re-enlistment greatly enhance the mean time to failure of the system. The goal of more than 20 years of MTTF could be easily achieved with computing node failure rate of 5×10^{-4}, or MTTF=2,000 hours. This means that, in ANTS, there is no need for ultra-dependable computing nodes. An inexpensive implementation is feasible.

Of course, for harsh environments such as military applications, where failure rate is high, ultra-dependable components are still required.

In [8], the Weibull distribution is recommended for its capability to include the aging effect in failure rate modeling. Even though we use constant failure rate (exponential distribution) in this analysis, we show that the ultra-dependability of ANTS was not dependent on the low failure rate of components. Of course, verification of these analysis results requires further investigation involved fault/error injection experiments. We are currently working on an emulation system of ANTS with fault/error injections from both hardware and software sources. Also, further work is needed to quantify our performance analysis.

Next, we show the two ANTS kernel implementations in a UNIX and Ethernet LAN environment. The first version will run as a UNIX application program while managing ANTS tasks. The second version is run as a UNIX daemon such that it uses virtually no computing resource while idle. In both case, the fault tolerant features are integrated to the kernel itself. In the fault injection experiments, we found that the fault tolerant features behave correctly as expected and that the system can indeed recover from even the most serious case of multiple computing node failures.

Finally, we presented results on realizing several well-known algorithms onto ANTS distributed computing environment. While most of the programming are in C or C++ language, we also successfully incorporate Matlab language in the ANTS environment. Since Matlab language has been widely used in digital signal processing profession, we expect many practical usages.

In summary, we have demonstrated the usefulness of the proposed ANTS concept. The dependability and performance goals are both achieved and shown. The practicality of the system is further enhanced by the interface to well known programming languages.

Acknowledgment

We would like to thank Patrick Dominic-Savio, Arun Paramadhathil, John Hammen, Tom Houweling, Brian Harrison, Brian Freburger and Nagendra Modadugu for their work and participation in this project.

References

[1] J. C. Lo, D. W. Tufts, and J. W. Cooley, "Active Nodal Task Seeking (ANTS): an approach to high-performance, ultra-dependable computing," *IEEE Trans. Aerospace and Electronic Syst.*, vol. 31, pp. 987–997, July 1995.

[2] K. Hwang and F. A. Briggs, *Computer Architecture and Parallel Processing.* McGraw-Hill Inc., 1984.

[3] H. Stone, *High-Performance Computer Architecture.* Addison-Wesley, 1987.

[4] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann Publishers Inc., Palo Alto, CA, 1990.

[5] AT&T, "Safeguard supplement," *The Bell System Tech. J.*, 1975.

[6] A. L. Hopkins, Jr., T. B. Smith III, and J. H. Lala, "FTMP - A highly reliable fault-tolerant multiprocessor for aircraft," *Proc. IEEE*, vol. 66, pp. 1221–1239, October 1978.

[7] J. H. Wensley, L. Lamport, J. Goldberg, M. W. Green, K. N. Levitt, P. M. Melliarsmith, R. E. Shostak, and C. B. Weinstock, "SIFT: Design and analysis of a fault-tolerant computer for aircraft control," *Proc. IEEE*, vol. 66, pp. 1240–1255, October 1978.

[8] D. P. Siewiorek and R. S. Swarz, *Reliable Computer Systems: Design and Evaluation.* 2nd ed., Digital Press, 1992.

[9] W. N. Toy, "Fault-tolerant design of local ESS processors," *Proc. IEEE*, vol. 66, pp. 1126–1145, October 1978.

[10] M. J. Iacoponi and D. K. Vail, "The fault tolerance approach of the advanced architecture on-board processor," in *Proc. 19th Int'l Symp. Fault-Tolerant Comput.*, pp. 6–12, June 1989.

[11] M. J. Iacoponi and S. F. McDonald, "Distributed reconfiguration and recovery in the advanced architecture on-board processor," in *Proc. 21st Int'l Symp. Fault-Tolerant Comput.*, pp. 436–443, June 1991.

[12] F. Cristian, B. Dancey, and J. Dehn, "Fault-tolerant in the advanced automation system," in *Proc. 20th Int'l Symp. Fault-Tolerant Computing*, pp. 6–17, June 1990.

[13] T. R. Dilenno, D. A. Yaskin, and J. H. Barton, "Fault-tolerant testing in the advanced automation system," in *Proc. 21st Int'l Symp. Fault-Tolerant Comput.*, pp. 18–25, June 1991.

[14] T. Takano, T. Yamada, K. Shutoh, and N. Kanekawa, "Fault-tolerant experiments of the "Hiten" onboard space computer," in *Proc. 21st Int'l Symp. Fault-Tolerant Comput.*, pp. 26–33, June 1991.

[15] M. Chereque, D. Powell, P. Reynier, J.-L. Richier, and J. Voiron, "Active replication in Delta-4," in *Proc. 22nd Int'l. Symp. Fault-Tolerant Comput.*, pp. 28–37, July 1992.

[16] E. N. Elnozahy and W. Zwaenepoel, "Replicated distributed processes in Manetho," in *Proc. 22nd Int'l Symp. Fault-Tolerant Computing*, pp. 18–27, July 1992.

[17] N. G. Shivaratri, P. Krueger, and M. Singhal, "Load distributing for locally distributed systems," *IEEE Computer*, vol. 25, pp. 33–44, December 1992.

[18] M. Linvy and M. Melman, "Load balancing in homogeneous broadcasting distributed systems," in *Proc.* ACM *Comput. Network Perf. Symp.*, 1982.

[19] J. E. Bahr, S. B. Levenstein, L. A. McMahon, J. T. Mullins, and A. H. Wottreng, "Architecture, design, and performance of Application System/400 (AS/400) multiprocessors," *IBM J. Res. Develop.*, vol. 36, pp. 1001–1013, November 1992.

[20] N. Carriero, E. Freeman, D. Gelernter, and D. Kaminsky, "Adaptive parallelism and Piranha," *IEEE Computer*, vol. 28, pp. 40–49, Jan. 1995.

[21] R. D. Blumofe and D. S. Park, "Scheduling large-scale parallel computations on networks of workstations," in *Proc. 3rd Int'l Symp. High-Performance Dist. Comput.*, pp. 96–105, August 1994.

[22] L. Cheng and A. Avizienis, "N-version programming: A fault tolerance approach to reliability of software operation," in *Proc. 8th Int'l. Symp. Fault-Tolerant Comput.*, pp. 3–9, June 1978.

[23] J. R. Sklaroff, "Redundancy management technique for space shuttle computers," *IBM I. Res. Develop.*, pp. 20–27, January 1976.

[24] Y. Savaria, N. C. Rumin, J. F. Hayes, and V. K. Agarwal, "Soft-error filtering: A solution to the reliability problem of future VLSI digital circuits," *Proc. IEEE*, vol. 74, pp. 669–683, May 1984.

[25] K. S. Trivedi, *Probability & Statistics with Reliability, Queuing, and Computer Science Applications.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1982.

[26] J. E. Angus, "On computing MTBF for a k-out-of-n:G repairable system," *IEEE Trans. Reliability*, vol. 37, pp. 312–313, August 1988.

[27] D. E. Comer and D. L. Stevens, *Internetworking with TCP/IP*, vol. 1. Prentice-Hall, NJ, 1988.

[28] R. P. Bianchini and R. W. Buskens, "Implementation of on-line distributed system-level diagnosis theory," *IEEE Trans. Comput.*, vol. 41, pp. 616–626, May 1992.

[29] W. W. Chu, L. J. Holloway, M. T. Lan, and K. Efe, "Task allocation in distributed data processing," *IEEE Computer*, pp. 57–69, November 1980.

[30] B. Harrison, R. Vaccaro, and D. Tufts, "Robust source localization in an acoustic waveguide," *to appear in J. Acoust. Soc. Amer.*, 1996.

[31] B. Harrison, R. Vaccaro, and D. Tufts, "Matched-field localization in uncertain ocean environments using propagation model families," *submitted to J. Acoust. Soc. Amer.*, 1996.

Table 1: Reconfiguration times for multiple *isolated* faults

Number of Isolated Faults	Reconfiguration Time(in Seconds)
1	2
2	3
3	3
4	4
5	4
6	5

Table 2: Reconfiguration times for multiple *chained* faults

Number of Isolated Faults	Reconfiguration Time(in Seconds)
1	N/A
2	5
3	15
4	17
5	21
6	28

Table 3. Task Execution Times of the RSWP Matlab program.

Task	Time (seconds)
1	5.2
2	23.1
3	29.3
4	36.6
5	43.2
6	50.0
7	50.5
8	50.6
9	51.7
10	51.0
11	50.1
12	50.9

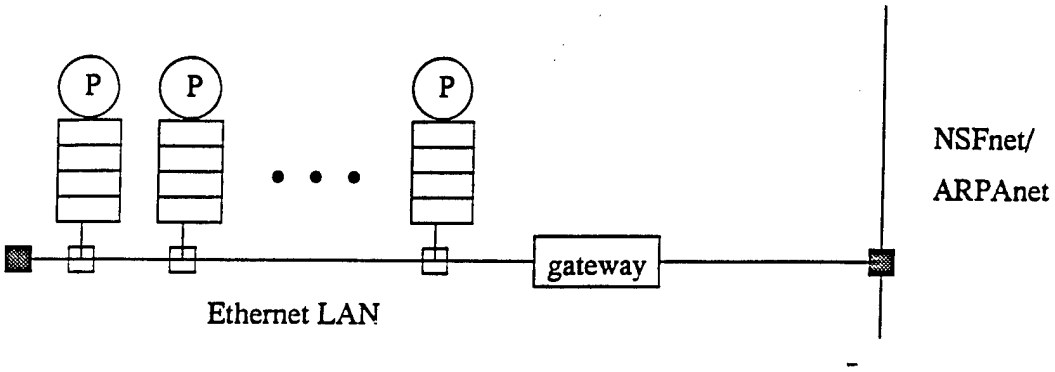

Figure 1: A network of workstations.

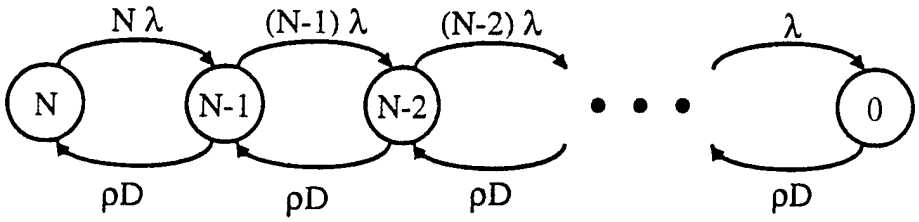

Figure 2: ANTS Dependability Model.

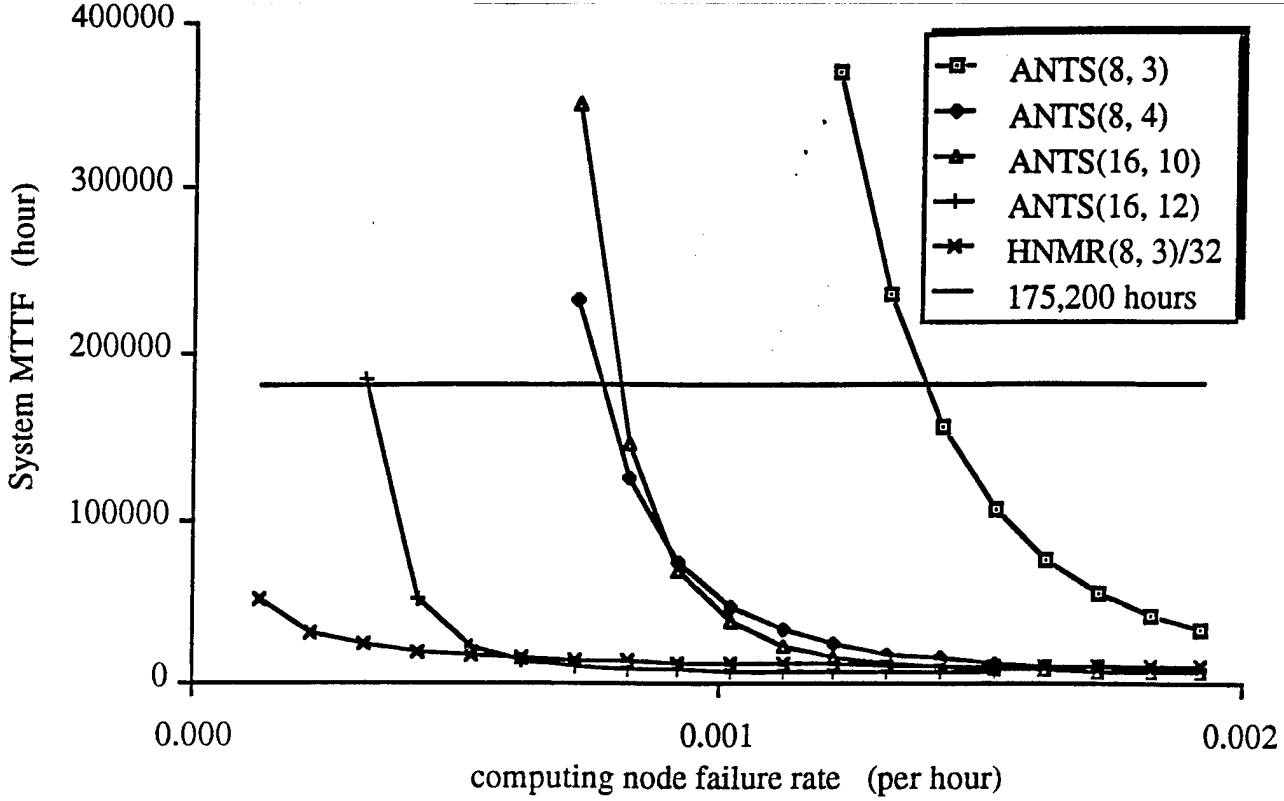

Figure 3: MTTF of ANTS for bus failure rate of 10^{-5} per hour.

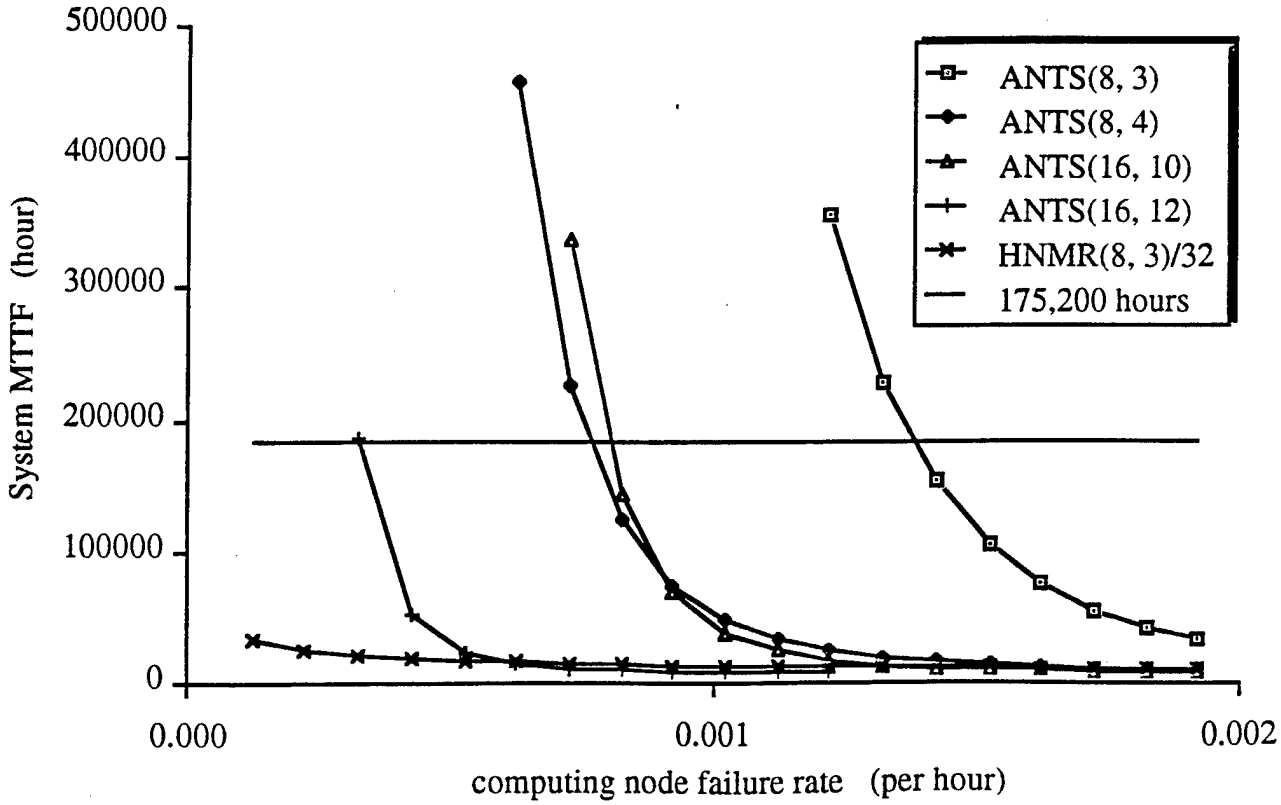

Figure 4: MTTF of ANTS for bus failure rate of 10^{-4} per hour.

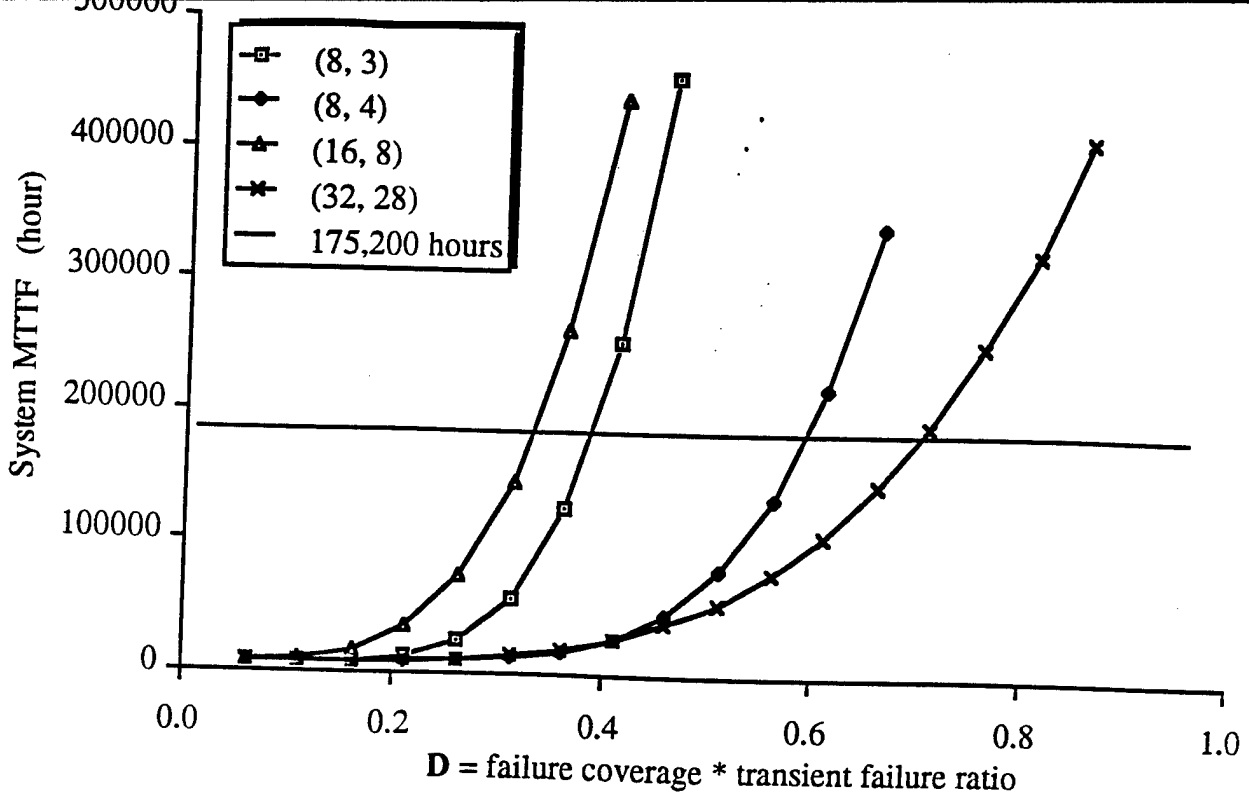

Figure 5: MTTF of ANTS for different diagnosis rate D and λ_B=10^{-5} per hour.

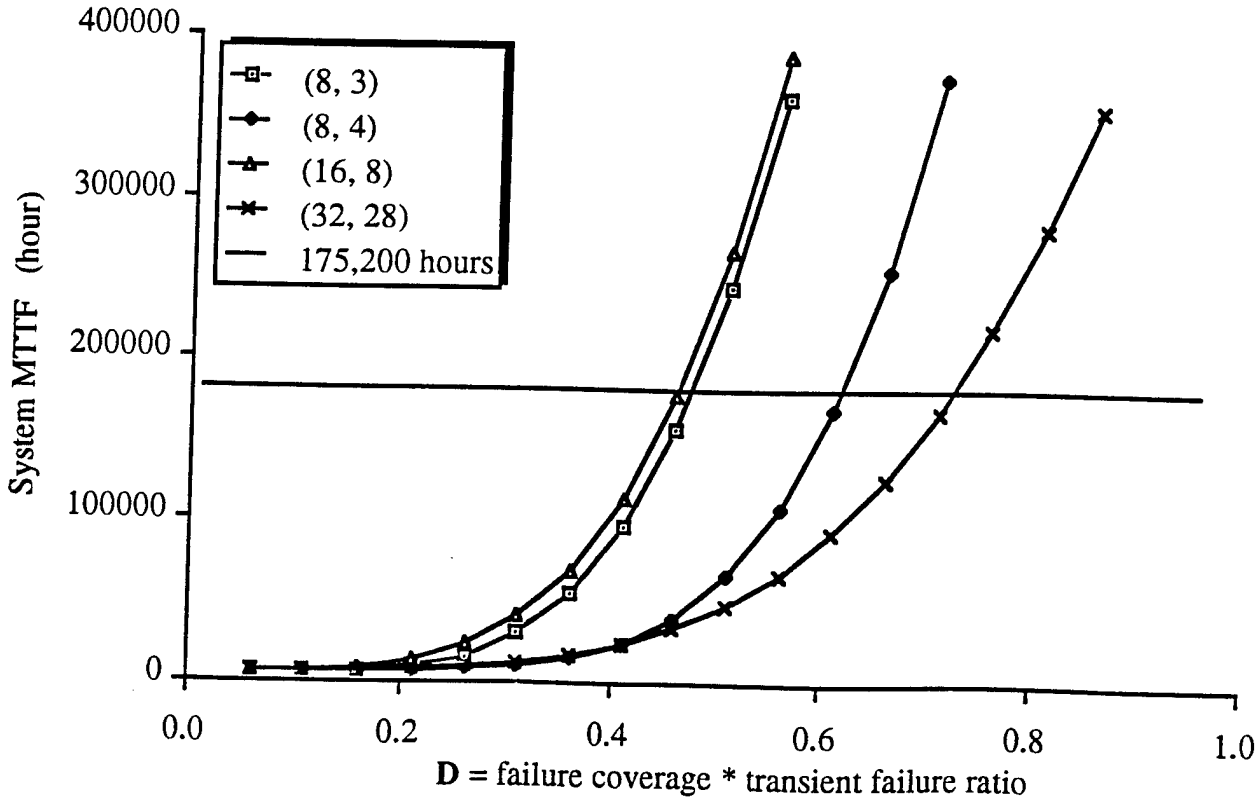

Figure 6: MTTF of ANTS for different diagnosis rate D and λ_B=10^{-4} per hour.

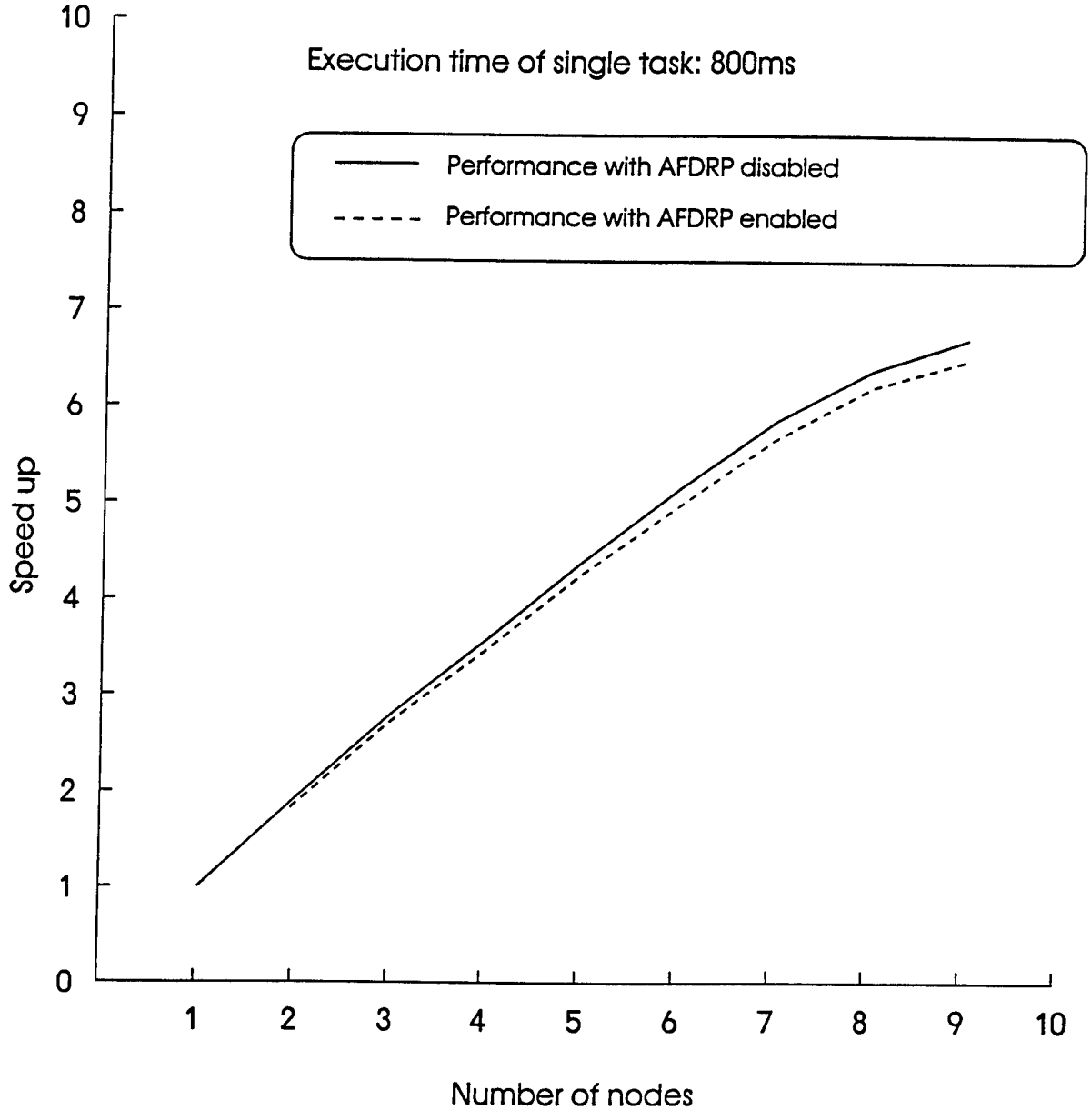

Figure 7: ANTS Speed-up factors comparison with and without fault-tolerant features.

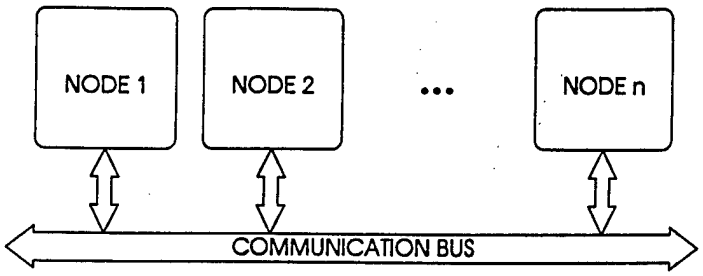

Figure 8: Architecure of the ANTS experimental system.

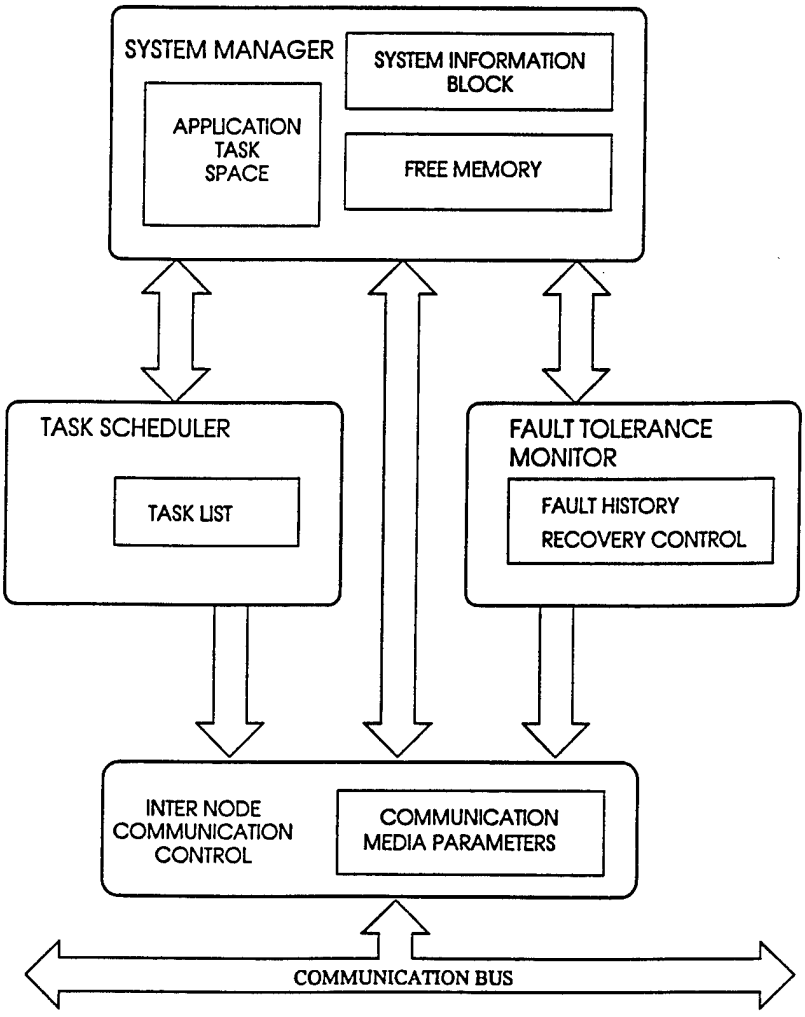

Figure 9: Overview of ANTS Kernel.

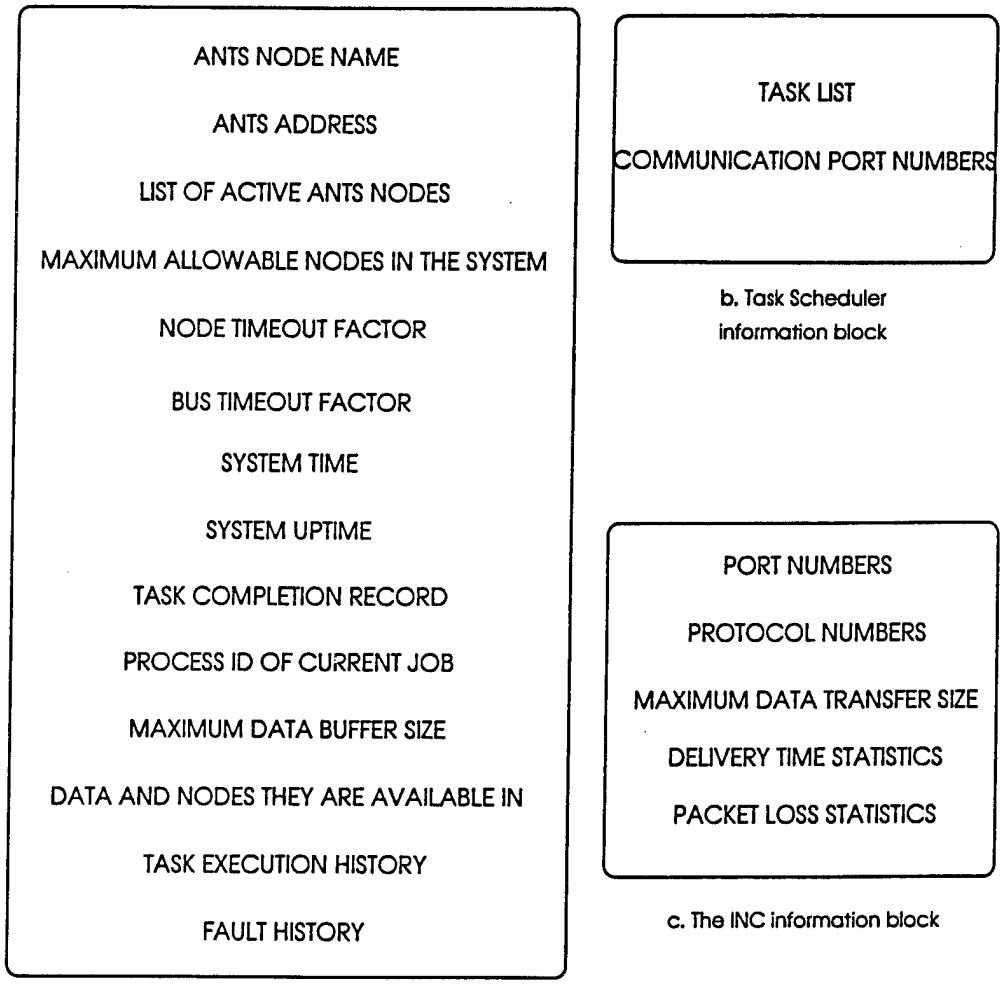

Figure 10: The system information block (SIB).

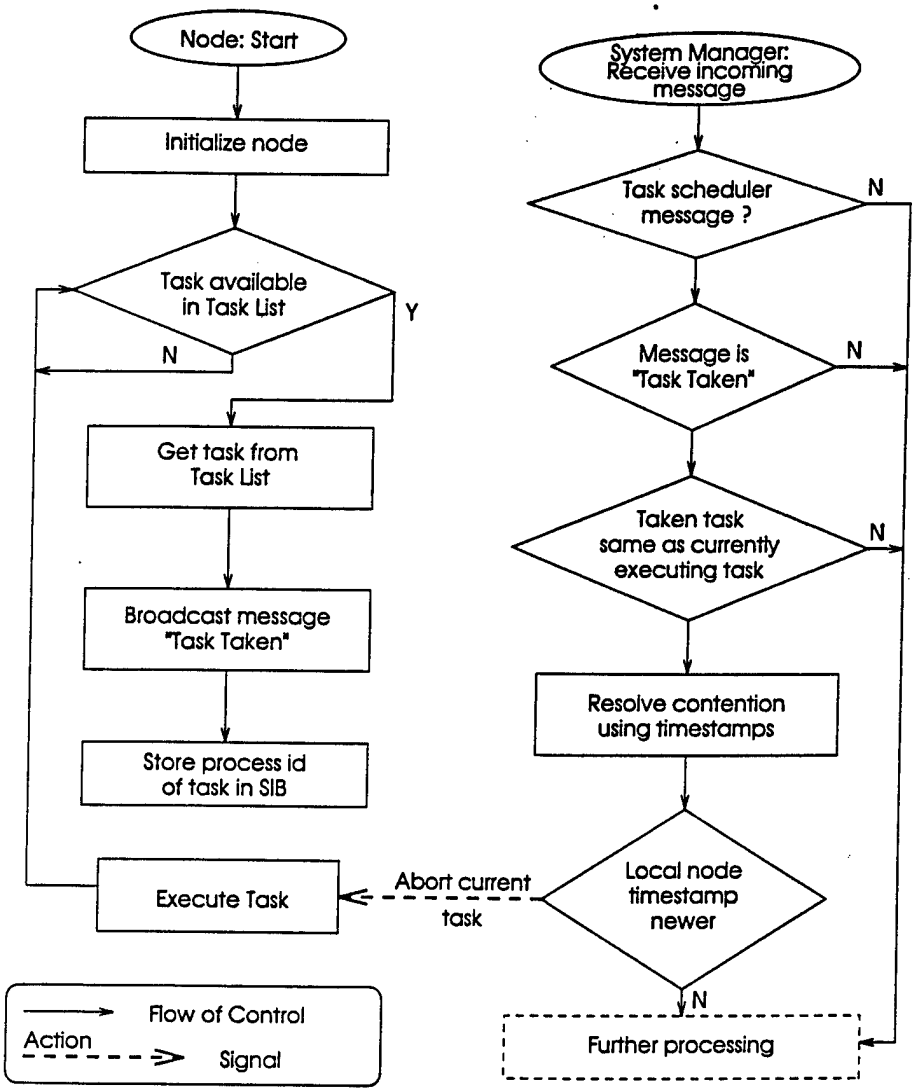

Figure 11: Contention resolution in ANTS task scheduler.

Task Scheduler	New Task	0 0 0 0 1 0 1 0
	Delete Task	0 0 0 1 0 1 0 0
System Manager	Request for Data	0 0 0 1 1 1 1 0
	Data	0 0 1 0 0 1 0 1
Fault Tolerance Monitor	Request for monitor report	0 0 1 0 1 0 0 0
	Monitor Report	0 0 1 1 0 0 1 0
	Remove Nodes	0 0 1 1 1 1 0 0
		MSb LSb

Figure 12: Bit pattern for the first byte of an ANTS message.

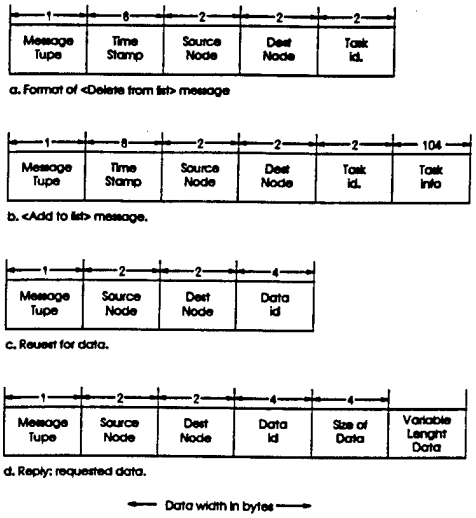

Figure 13: Message formats for task scheduler and data transfer.

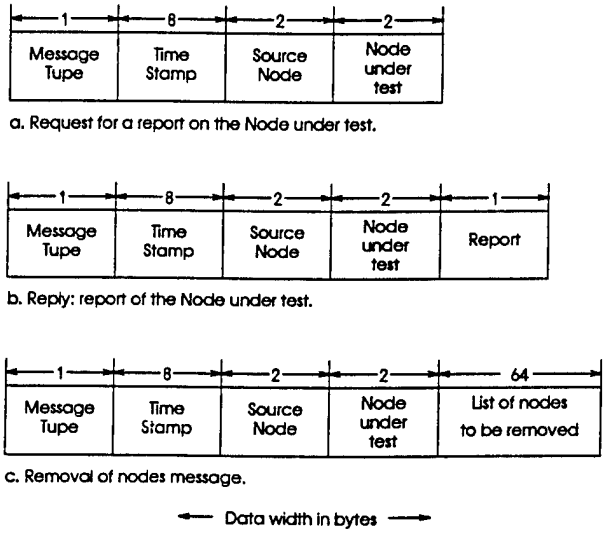

Figure 14: Message formats for AFDRP.

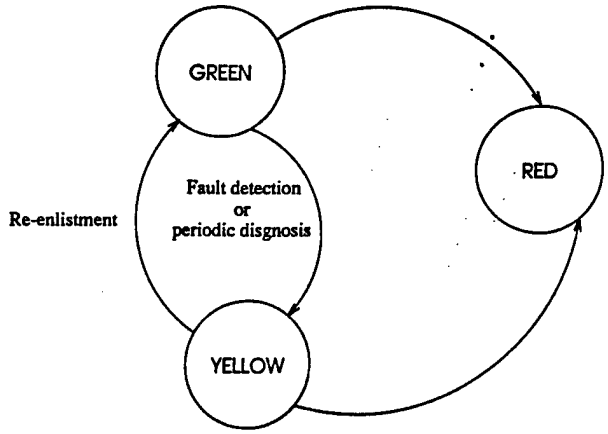

Figure 15: Run-time partitioning of fault-tolerant ANTS system.

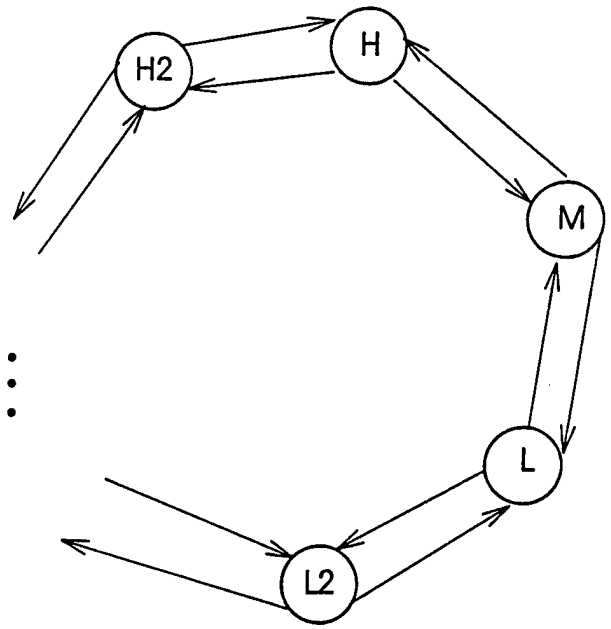

Figure 16: Monitoring for faulty ANT nodes.

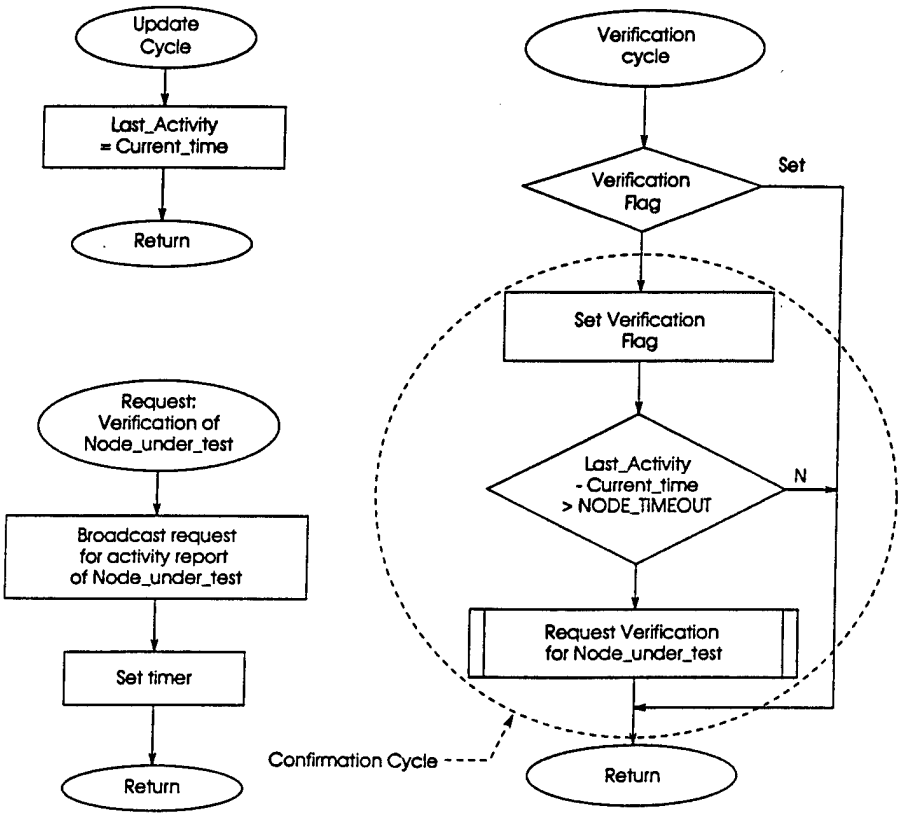

Figure 17: AFDRP: update and verify cycles.

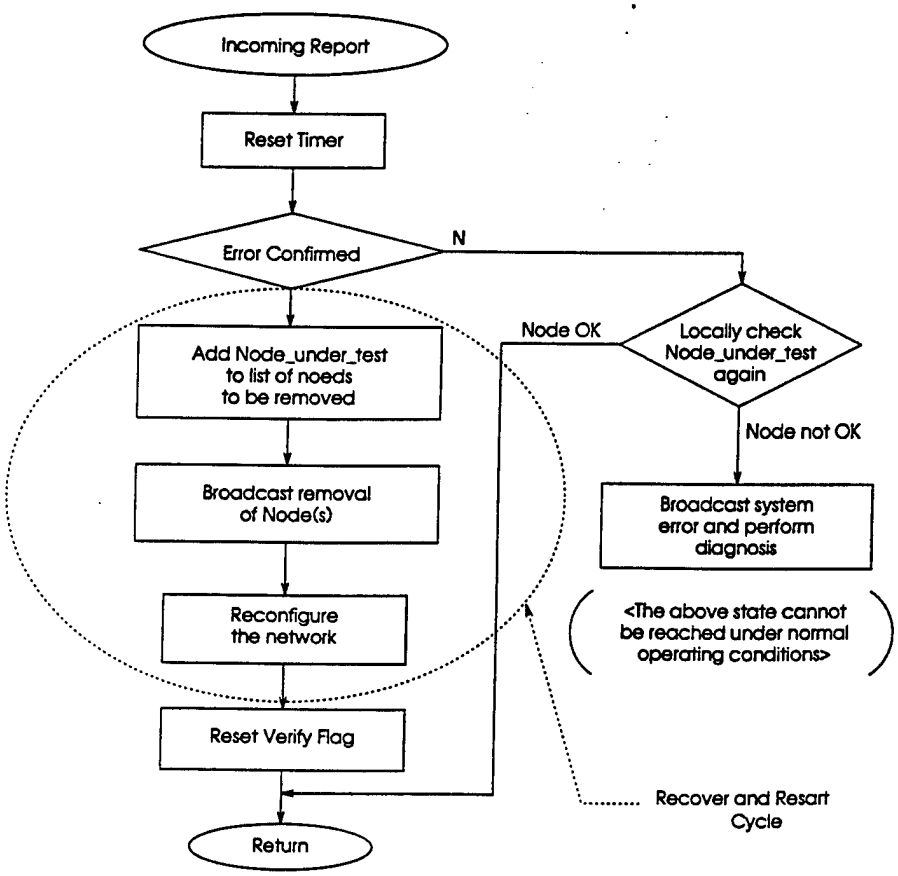

Figure 18: AFDRP: recovery and restart cycles.

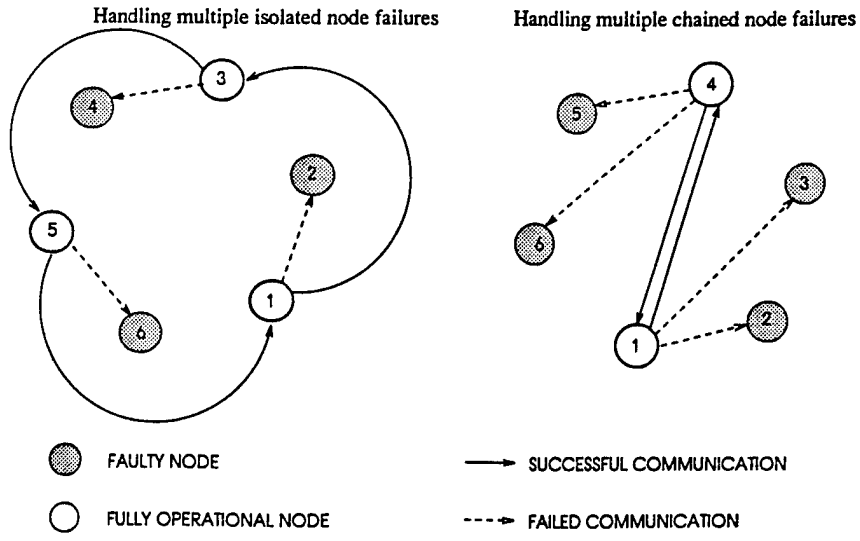

Figure 19: Detection of multiple isolated and multiple chained node failures.

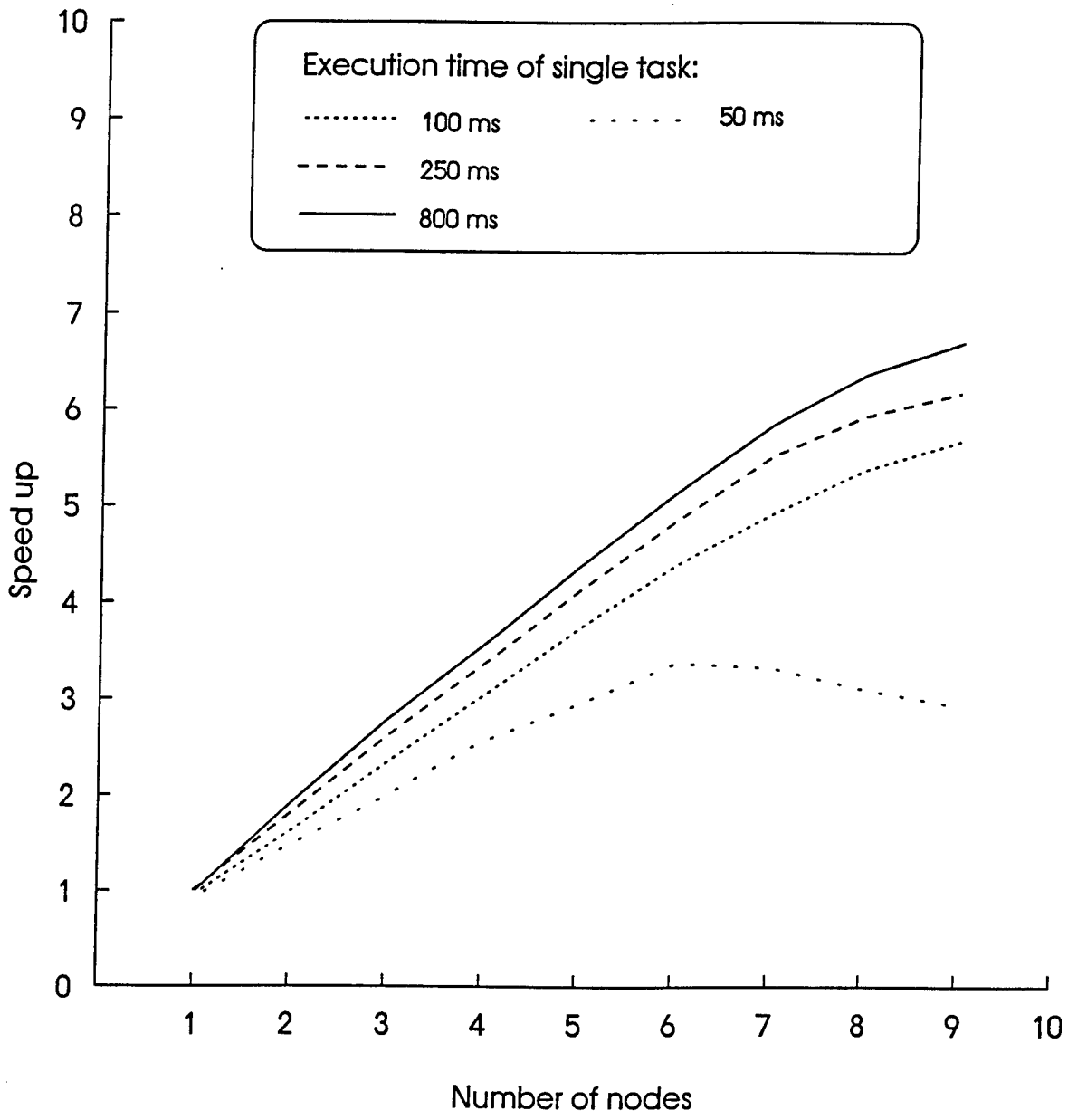

Figure 20: Speed-up factors of ANTS.

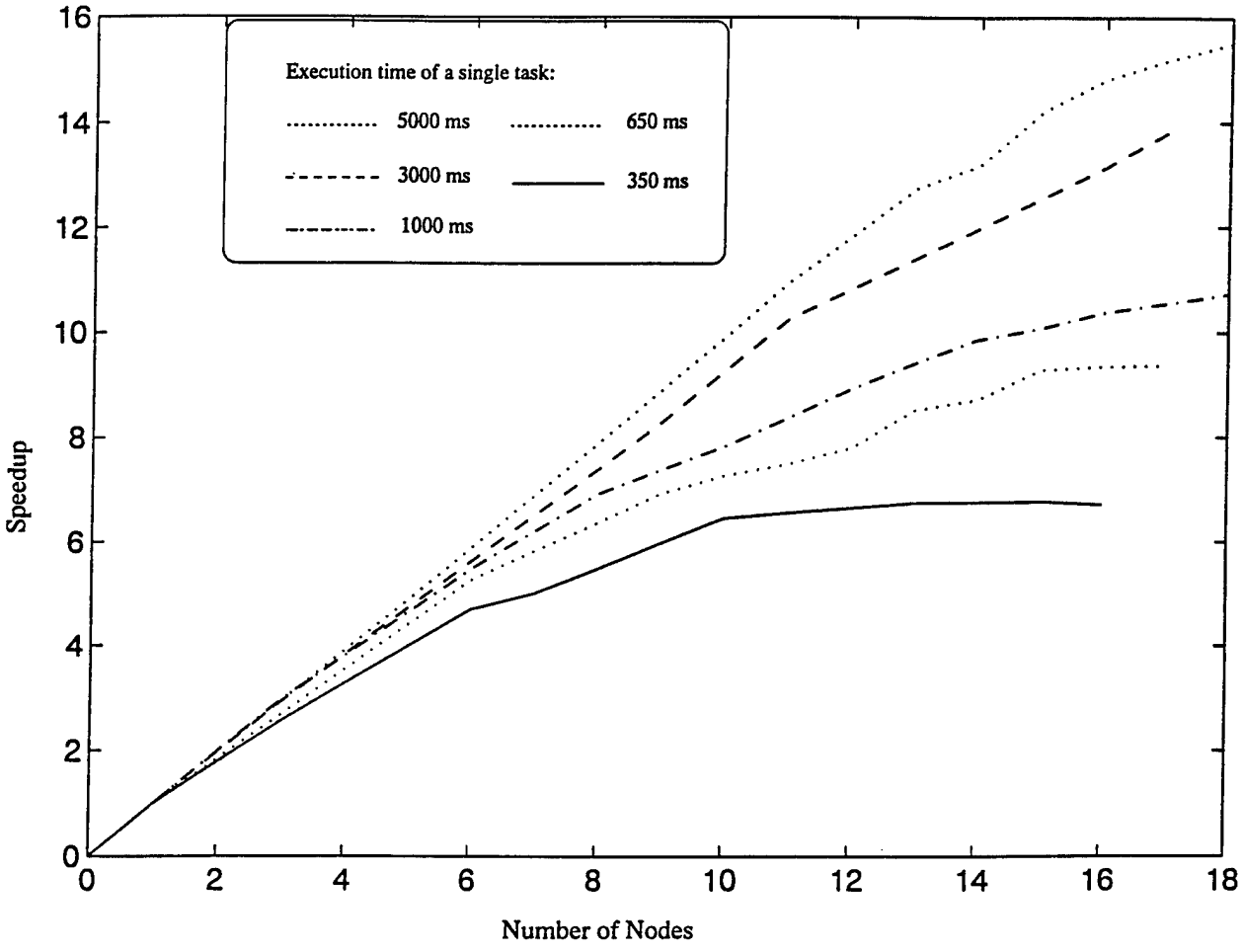

Figure 21: Speed-up factors of ANTS with extended average task execution times.

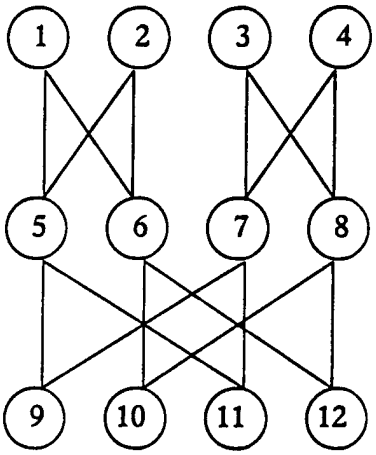

Figure 22: Data dependency graph of an example job.

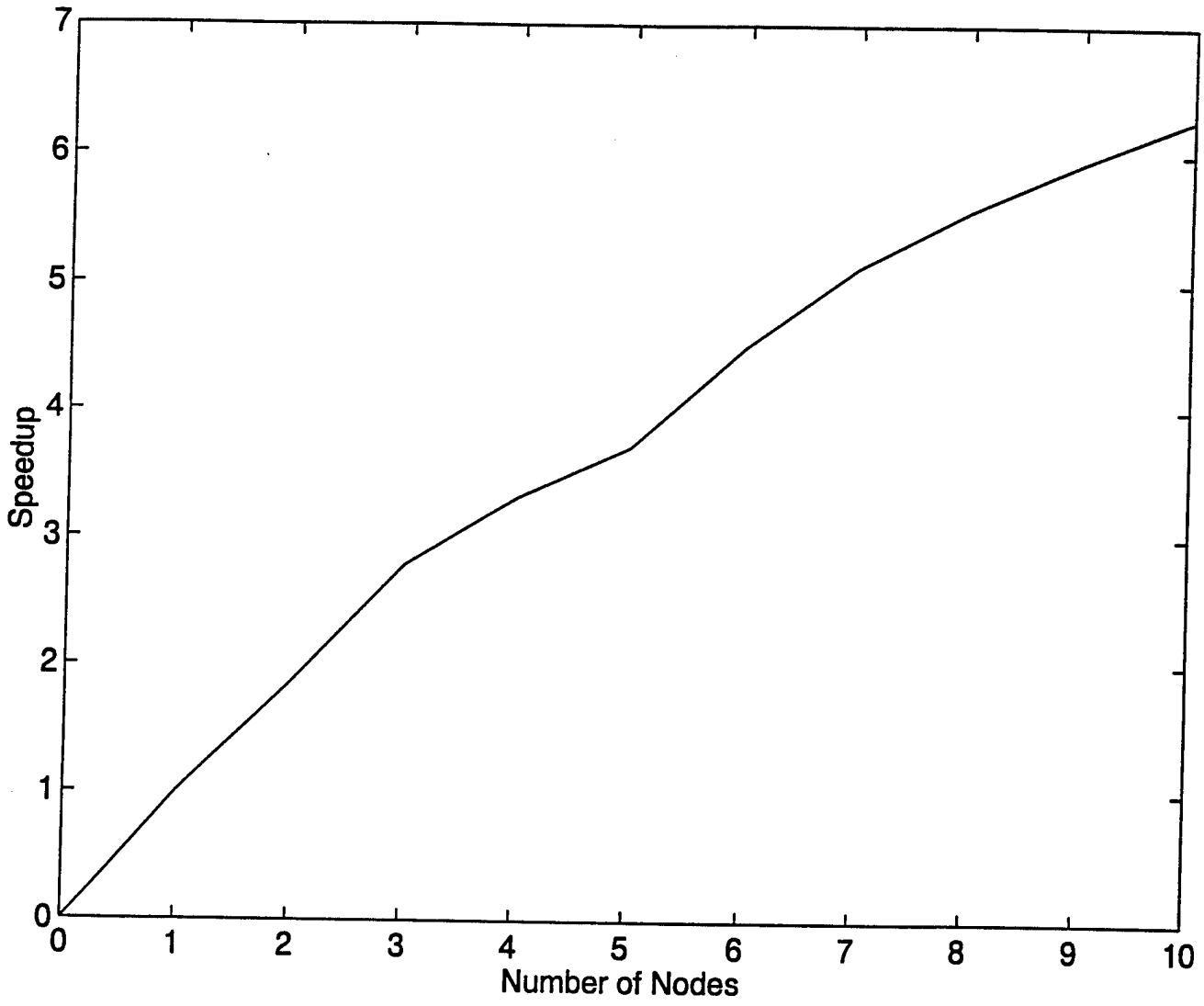

Figure 23: Speed-up factors of RSWP algorithm on ANTS with Matlab interface.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	8/20/96	Final, 1/15/94 - 5/24/96

4. TITLE AND SUBTITLE
Ultra-Dependable, High-Performance, Real Time Signal Processing

5. FUNDING NUMBERS
N00014-94-1-0479

6. AUTHOR(S)
Jien-Chung Lo
Donald W. Tufts
James W. Cooley

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Department of Electrical & Computer Engineering
University of Rhode Island
Kingston, RI 02881-0805

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Office of Naval Research
Ballston Tower One
800 North Quincy St.
Arlington, VA 22217-5660

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The Active Nodal Task Seeking (ANTS) approach to the design of multicomputer systems is named for its basic component: an Active Nodal Task-Seeker (ANT). An ANTS multicomputer system can achieve a mean time to failure of more than 20 years.

In this final report, we present the many findings during the project duration. First, the first version of ANTS kernel has been implemented on a network of UNIV workstations. Based on these implementations, we have successfully demonstrated the proposed fault tolerant techniques via fault injection experiments. The system detects faults and recover itself as expected within seconds. The dependability goal predicted in the theoretical model is validated. We then program and execute several well-known algorithms in ANTS distributed computing environment to demonstrate the high-performance features.

14. SUBJECT TERMS
Dependable distributed systems, fault-tolerant computing, high-performance signal processing, real-time applications

15. NUMBER OF PAGES
61

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
unclassified	unclassified	unclassified	UL

AD NUMBER	DATE	DTIC ACCESSION NOTICE

1. REPORT IDENTIFYING INFORMATION

A. ORIGINATING AGENCY

Office of Naval Research

B. REPORT TITLE AND/OR NUMBER

Ultra-Dependable, High-Performance, Real-Time Signal Processing

C. MONITOR REPORT NUMBER

D. PREPARED UNDER CONTRACT NUMBER

N00014-94-1-0479

2. DISTRIBUTION STATEMENT

Distribution Unlimited

19960906 011

address on

and 2.

ports

formation

1

2

DTIC FORM 50 FEB 86

PREVIOUS EDITIONS ARE OBSOLETE